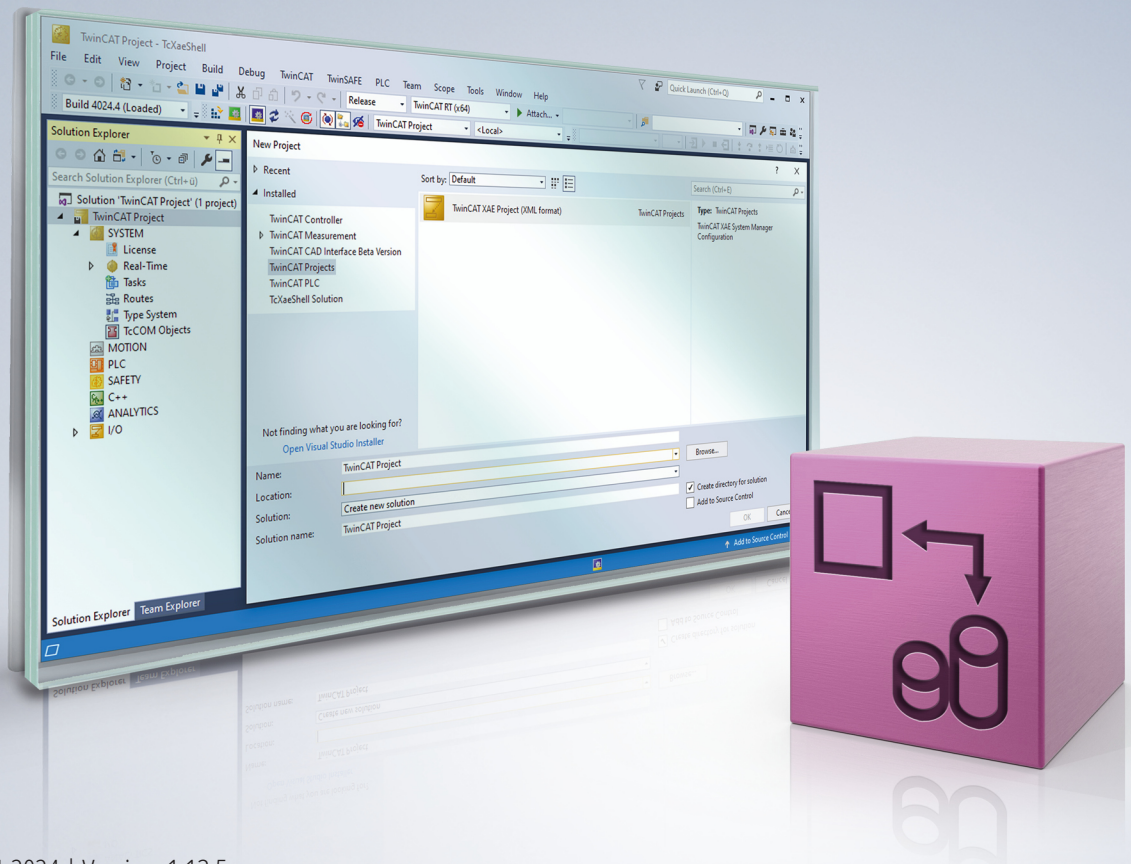
BECKHOFF New Automation Technology

Handbuch | DE

TF6420

TwinCAT 3 | Database Server



25.11.2024 | Version: 1.13.5

Inhaltsverzeichnis

1 Vorwort 5
1.1 Hinweise zur Dokumentation 5
1.2 Zu Ihrer Sicherheit 6
1.3 Hinweise zur Informationssicherheit 7
2 Übersicht 8
3 Installation 10
3.1 Systemvoraussetzungen 10
3.2 Installation 10
3.3 Lizenzierung 13
3.4 Installation Windows CE 15
3.5 Installation TwinCAT/BSD 17
4 Technische Einführung 18
4.1 Grundkonzept 18
4.2 Einsatzgebiete und Netzwerktopologien 20
4.3 Kompatibilität 21
5 Konfiguration 24
5.1 Konfigurator 24
5.1.1 Integration in Visual Studio 24
5.1.2 Standalone-Konfigurator 104
5.2 Datenbanken 126
5.2.1 Allgemeine Informationen 127
5.2.2 MS SQL Datenbank 129
5.2.3 MS SQL Compact Datenbank 131
5.2.4 MySQL Datenbank 132
5.2.5 Oracle Datenbank 134
5.2.6 SQLite 135
5.2.7 ASCII-File 137
5.2.8 XML-Database 138
5.2.9 ODBC Datenbanken 144
5.2.10 MS Access Datenbank 151
5.2.11 MS Excel Datenbank 152
5.2.12 MongoDB 153
5.2.13 PostgreSQL 155
5.2.14 InfluxDB 156
5.2.15 InfluxDB2 157
6 SPS-API 160
6.1 Tc3_Database 160
6.1.1 Funktionsbausteine 160
6.1.2 Datentypen 229
6.1.3 Globale Konstanten 249
6.1.4 Obsolete 250
6.2 Tc2_Database 295
6.2.1 Funktionsbausteine 297

6.2.2 Datentypen ... 325
6.2.3 Globale Konstanten ... 329

7 Beispiele ... 331
7.1 Tc3_Database ... 331
7.1.1 Szenario-Beispiele ... 331
7.1.2 Best Practises ... 351
7.2 Tc2_Database ... 364
7.2.1 Erstellen einer MS Access Datenbank ... 364
7.2.2 Starten / Stoppen zyklisches Loggen ... 367
7.2.3 Loggen einer SPS-Variablen mit FB_DBWrite ... 369
7.2.4 Beispiel mit dem FB_DBRecordInsert und FB_DBRecordSelect Baustein ... 373
7.2.5 Stored Procedures mit FB_DBStoredProceduresRecordArray ... 375
7.2.6 XML als Datenbank nutzen ... 378
7.2.7 XPath-Beispiel zur Darstellung der unterschiedlichen SELECT-Typen ... 383
7.2.8 XPath-Beispiel mit XML-Schema ... 386

8 Anhang ... 391
8.1 Rückgabewerte ... 391
8.1.1 Tc3_Database ... 391
8.1.2 Tc2_Database ... 401
8.2 FAQ - Häufig gestellte Fragen und Antworten ... 414
8.3 Support und Service ... 415

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

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

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Deutschland.
Weitergabe sowie Vervielfältigung dieses Dokuments, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet.
Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten.

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

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

Haftungsausschluss

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

Qualifikation des Personals

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

Signalwörter

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

Warnungen vor Personenschäden

⚠ GEFAHR

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

⚠ WARNUNG

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

⚠ VORSICHT

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

Warnung vor Umwelt- oder Sachschäden

HINWEIS

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

Information zum Umgang mit dem Produkt

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

1.3 Hinweise zur Informationssicherheit

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

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

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

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

2 Übersicht

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 große Bibliothek von SPS-Funktionsbausteinen für ein Maximum an Flexibilität. So können beispielsweise direkt aus der SPS SQL-Befehle wie Insert oder Select verwendet werden. Um die SPS bei Bedarf zu entlasten, besteht zusätzlich die Möglichkeit sogenannte gespeicherte Prozeduren (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.

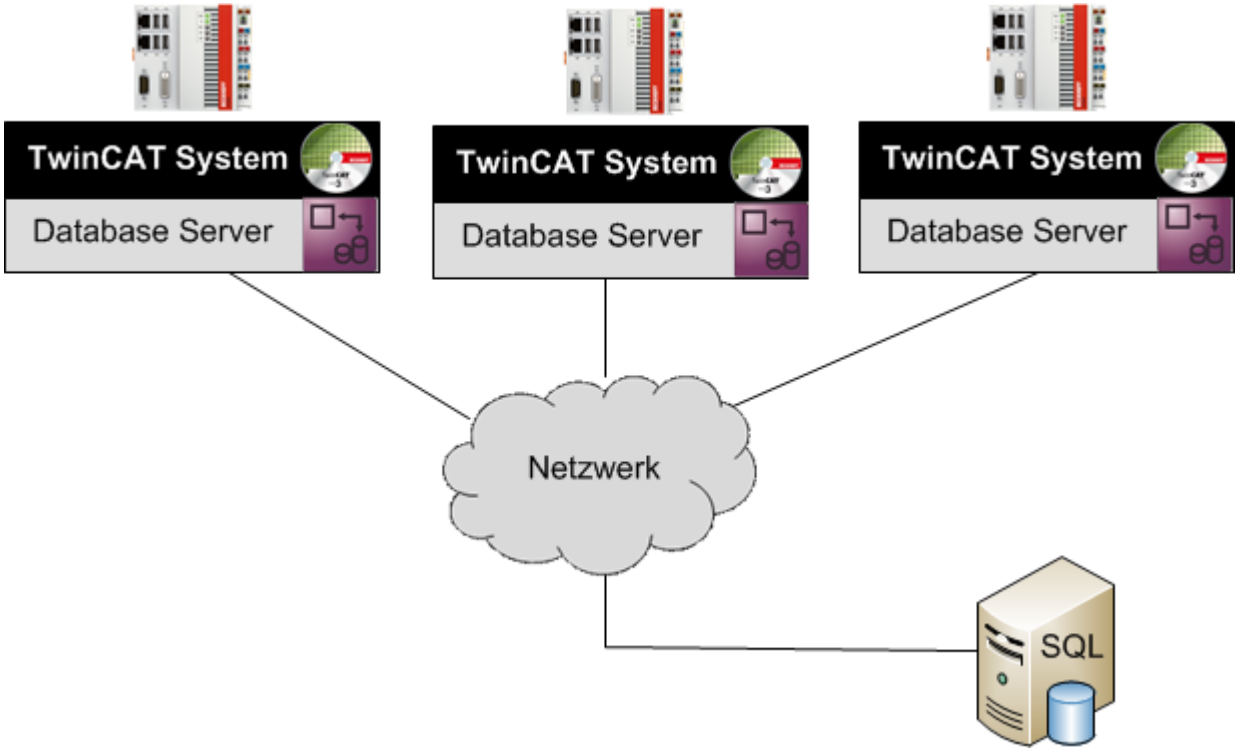
Der TwinCAT Database Server unterstützt viele verschiedene Datenbanksysteme: MS SQL, MS SQL Compact, MS Access, MySQL, PostgreSQL, DB2, Oracle, Interbase, Firebird, ASCII- (z. B. .txt oder .csv) sowie XML-Dateien. Neu dazugekommen sind die sogenannten NoSQL Datenbanken, mit der Unterstützung von MongoDB wird der Grundstein für diese Datenbank-Familie gelegt. (Siehe auch: Deklarieren der verschiedenen Datenbanktypen)

Komponenten

- TwinCAT Database Server [▶ 18]: Service, der mit TwinCAT zusammen gestartet und gestoppt wird. Er bildet das Bindeglied zwischen TwinCAT-System und Datenbank.
- Konfigurator [▶ 104]: 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 [▶ 295]: Die SPS-Bibliothek beinhaltet verschiedene Funktionsbausteine. Diese ermöglichen zum Beispiel das Herstellen einer Datenbankverbindung und die Erstellung einer neuen Tabelle sowie das Schreiben von Daten in völlig beliebige Tabellenstrukturen mithilfe von Insert-Befehlen und das Auslesen über Select-Befehle. Möglich ist auch das Aktualisieren und Löschen von Datenbankeinträgen sowie das Anstoßen von gespeicherten Prozeduren (Stored Procedures). NoSQL Datenbanken haben eigene Funktionsbausteine, die zum Beispiel für das Handling mit flexiblen JSON-Dokumenten in der SPS optimiert sind. Das Funktionsprinzip ist identisch.

Funktionsprinzip

Der Database Server kommuniziert innerhalb des TwinCAT-Systems über ADS. Nach außen verbindet er sich mit der jeweils konfigurierten Datenbank. Mögliche Netzwerk-Topologien finden Sie im Abschnitt „Einsatzgebiete und Netzwerktechnologien [▶ 20]“.


TwinCAT System
Database Server
TwinCAT System
Database Server
TwinCAT System
Database Server
Netzwerk
SQL

3 Installation

3.1 Systemvoraussetzungen

Technische Daten	TF6420 TwinCAT 3 Database Server
Zielsysteme	Windows 10, WinCE, TwinCAT/BSD x86, x64 und ARM
.NET Framework	.Net 4.5.1 oder höher WinCE: .NET 3.5
Min. TwinCAT-Version	3.1.4018
Min. TwinCAT-Level	TC1200 | TwinCAT 3 PLC

3.2 Installation

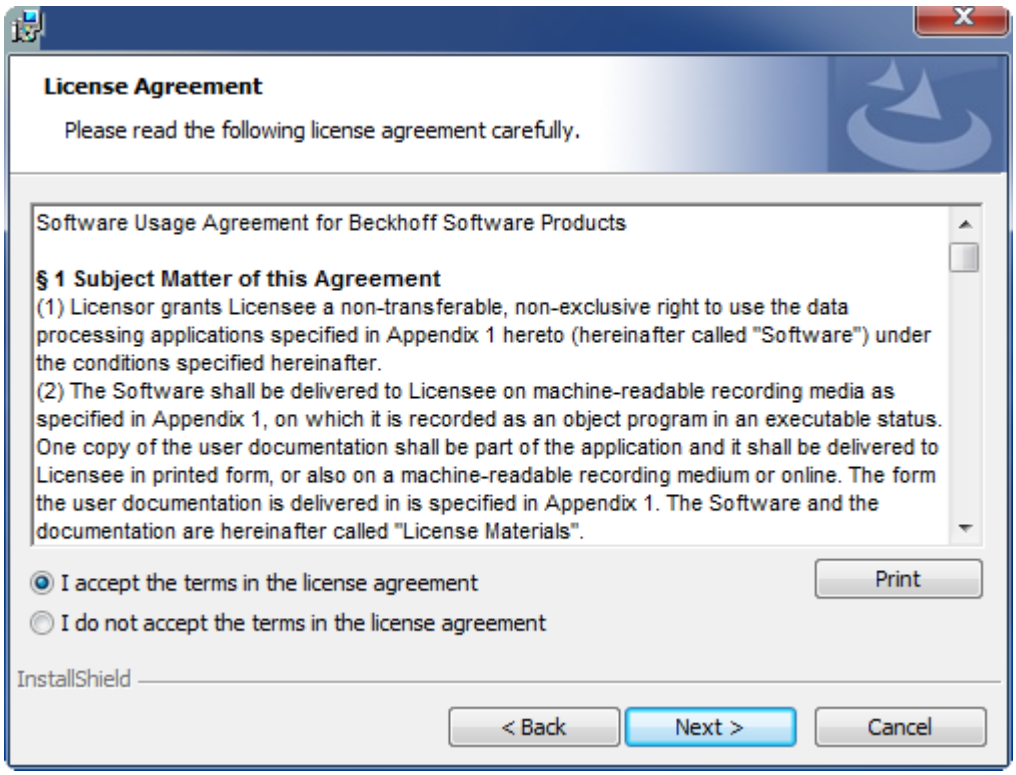
Nachfolgend wird beschrieben, wie die TwinCAT 3 Function für Windows-basierte Betriebssysteme installiert wird.

✓ Die Setup-Datei der TwinCAT 3 Function wurde von der Beckhoff-Homepage heruntergeladen.

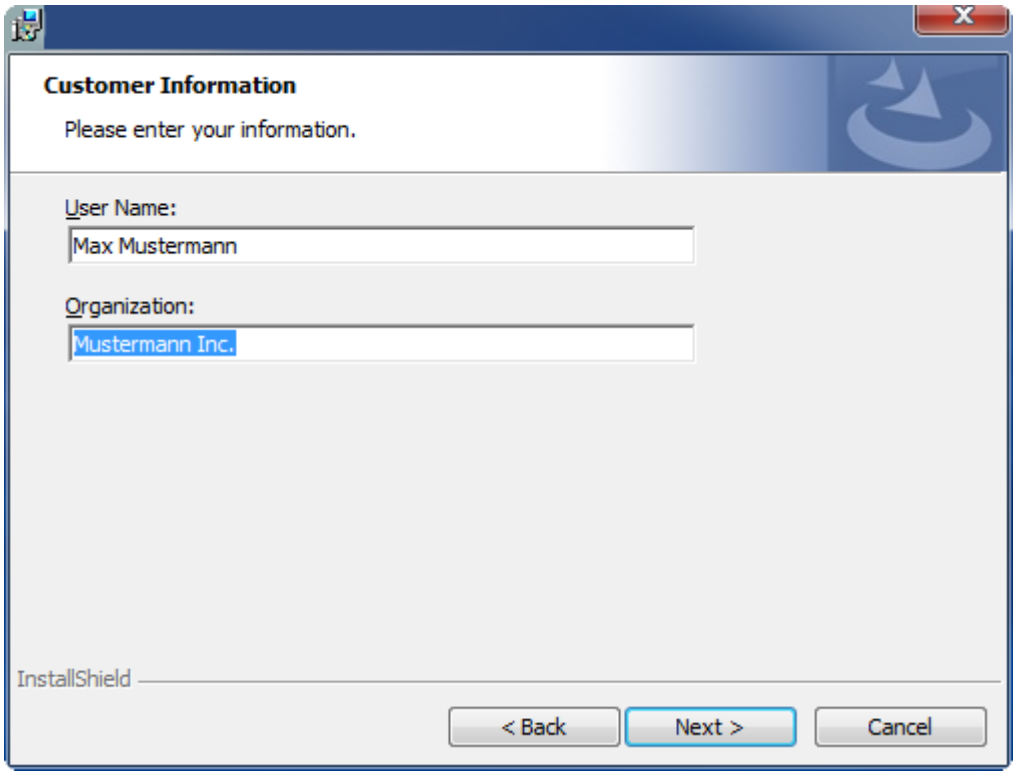
1. Führen Sie die Setup-Datei als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.

   ⇨ Der Installationsdialog öffnet sich.

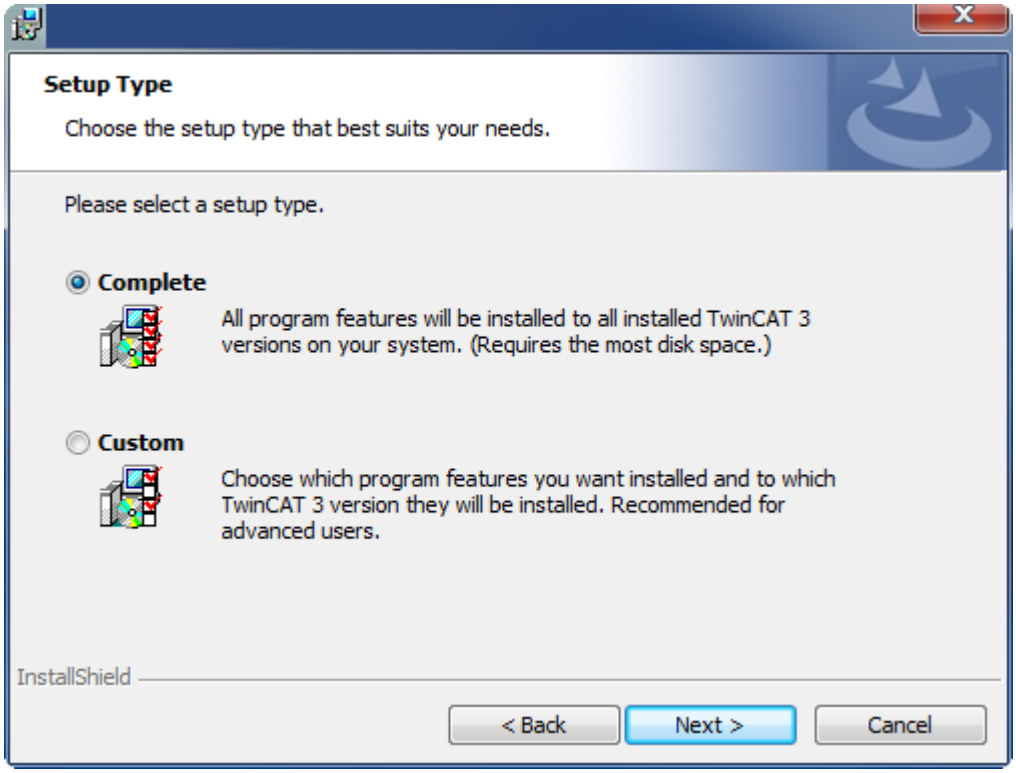
2. Akzeptieren Sie die Endbenutzerbedingungen und klicken Sie auf **Next**.



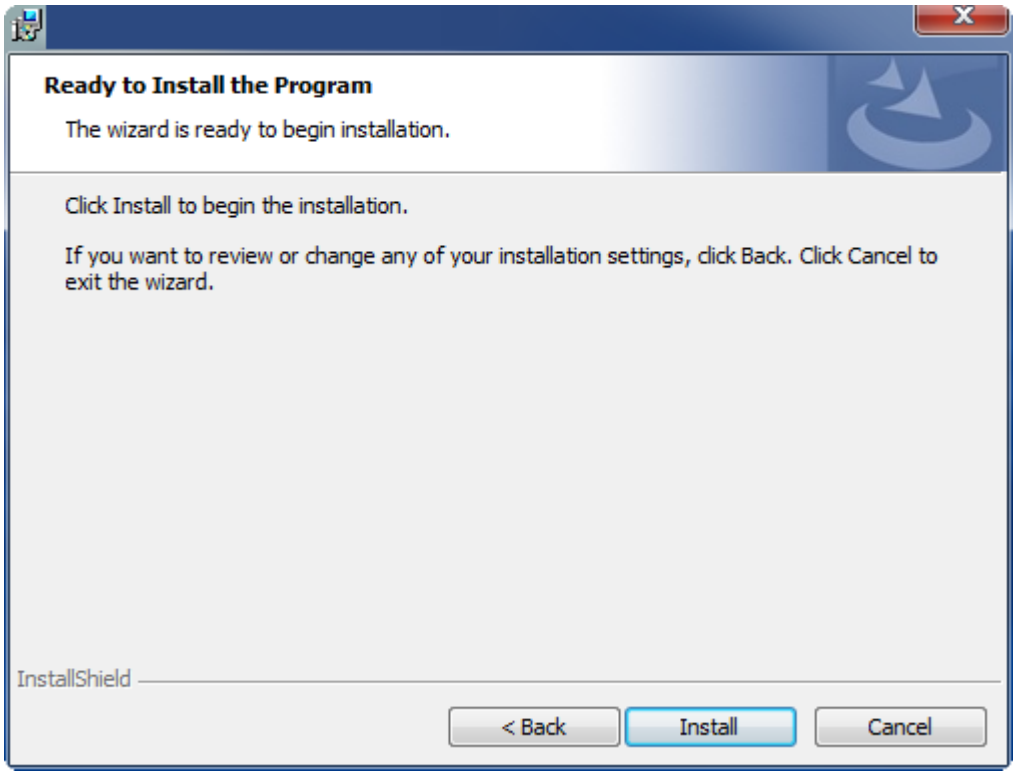
3. Geben Sie Ihre Benutzerdaten ein.



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

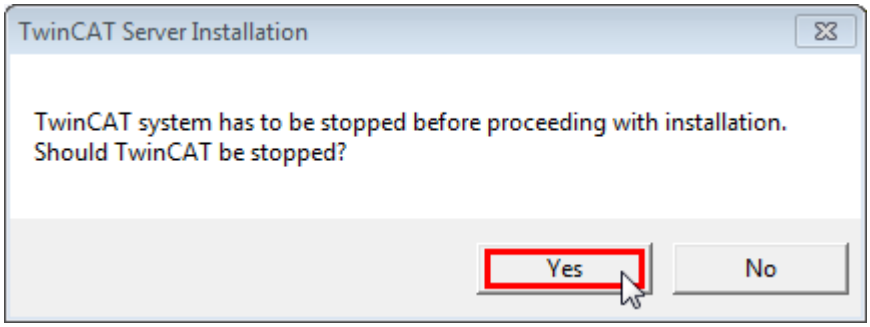


5. Wählen Sie **Next** und anschließend **Install**, um die Installation zu beginnen.

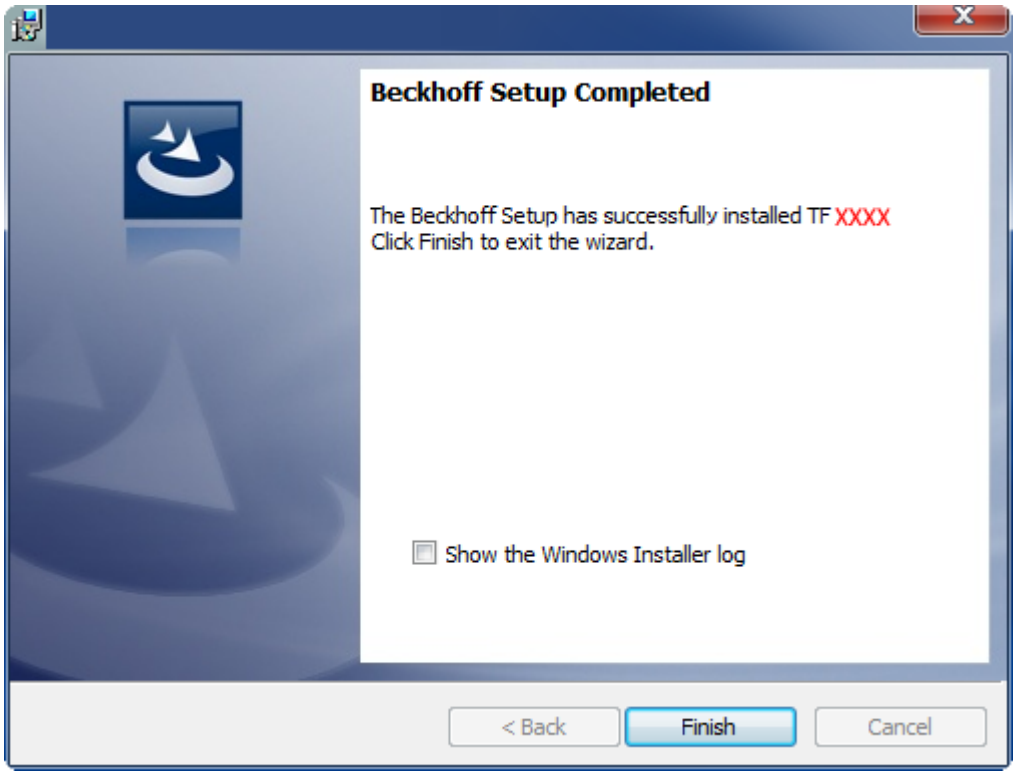


⇨ Ein Dialog weist Sie darauf hin, dass das TwinCAT-System für die weitere Installation gestoppt werden muss.

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



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



⇨ Die TwinCAT 3 Function wurde erfolgreich installiert.

3.3 Lizenzierung

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

Lizenzierung der Vollversion einer TwinCAT 3 Function

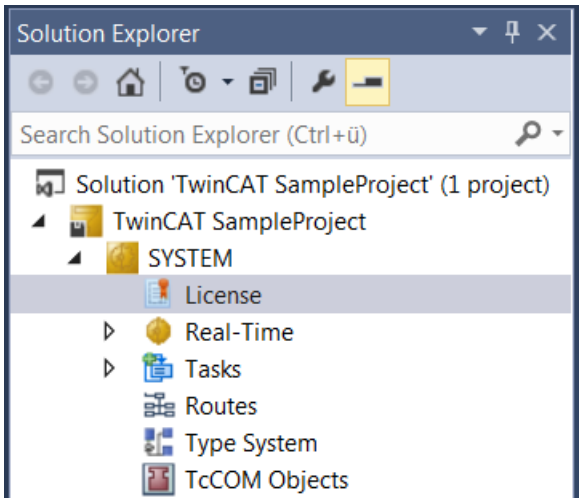
Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT-3-Lizenzierung“.

Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

Eine 7-Tage-Testversion kann nicht für einen TwinCAT-3-Lizenz-Dongle freigeschaltet werden.

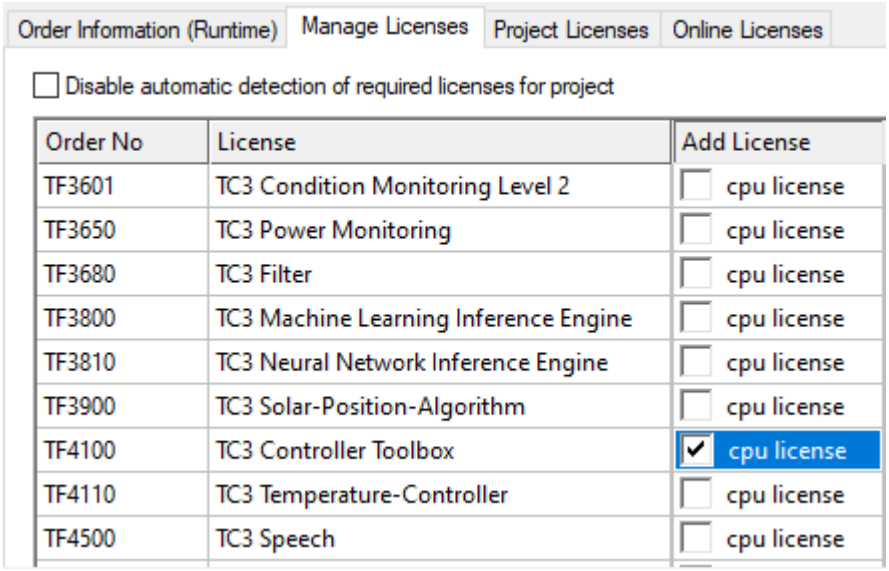
1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.
3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste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

4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**



⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.

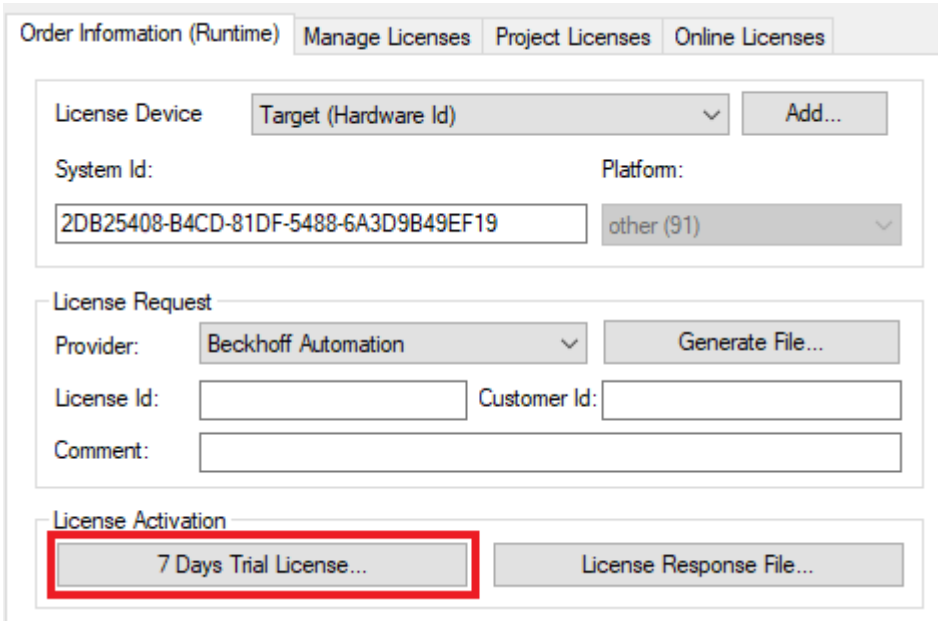
5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).



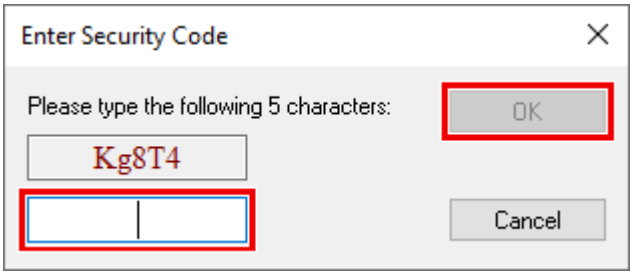
6. Öffnen Sie die Registerkarte **Order Information (Runtime).**

⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt.

7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



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



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.
   ⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

3.4 Installation Windows CE

Nachfolgend wird beschrieben, wie eine TwinCAT 3 Function (TFxxx) auf einem Beckhoff Embedded-PC mit Windows CE installiert wird.

1. Download der Setup-Datei und Installation [▶ 15]
2. CAB-Datei auf das Windows-CE-Gerät übertragen [▶ 16]
3. CAB-Datei auf dem Windows-CE-Gerät ausführen [▶ 16]

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

- Upgrade der Software [▶ 16]

Download der Setup-Datei und Installation

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

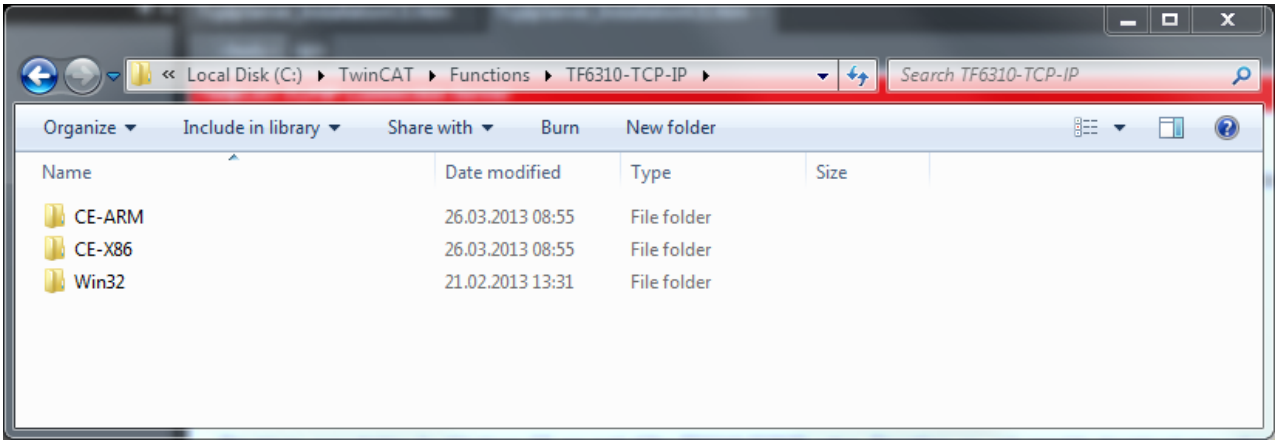
Laden Sie das TFxxx-Setup herunter und installieren Sie die TwinCAT 3 Function wie es im Abschnitt Installation [▶ 10] beschrieben wird.

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

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

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

Beispiel: Installationsordner „TF6310“



CAB-Datei auf das Windows-CE-Gerät übertragen

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

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

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

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

CAB-Datei auf dem Windows-CE-Gerät ausführen

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

Nach dem Neustart des Gerätes werden die Dateien der TwinCAT 3 Function (TFxxxx) automatisch im Hintergrund geladen und sind verfügbar.

Die Software wird in dem folgenden Verzeichnis auf dem Windows-CE-Gerät installiert:
\Hard Disk\TwinCAT\Functions\TFxxxx

Upgrade der Software

Wenn auf dem Windows-CE-Gerät bereits eine ältere Version der TwinCAT 3 Function installiert ist, führen Sie die folgenden Schritte auf dem Windows-CE-Gerät durch, um ein Upgrade auf eine neue Version durchzuführen:

1. Öffnen Sie den CE Explorer, indem Sie auf **Start > Run** klicken und „Explorer“ eingeben.

2. Navigieren Sie nach *\Hard Disk\TwinCAT\Functions\TFxxx\xxxx*.
3. Benennen Sie die Datei *Tc*.exe* in *Tc*.old* um.
4. Starten Sie das Windows-CE-Gerät neu.
5. Übertragen Sie die neue CAB-Datei auf das Windows-CE-Gerät.
6. Führen Sie die CAB-Datei auf dem Windows-CE-Gerät aus und installieren Sie die neue Version.
7. Löschen Sie die Datei *Tc*.old*.
8. Starten Sie das Windows-CE-Gerät neu.

⇨ Nach dem Neustart ist die neue Version aktiv.

3.5 Installation TwinCAT/BSD

Der TwinCAT 3 Database Server steht als Package für TwinCAT/BSD im Package-Repository bereit. Unter dem Package-Namen „TF6420-Database-Server“ kann er über den Befehl

```
doas pkg install TF6420-Database-Server
```

installiert werden.

Weitere Informationen über den Package Server finden Sie im Bereich Embedded-PC im TwinCAT/BSD-Handbuch.

Nach einem Neustart des Systems oder Restart von TwinCAT, wird der TwinCAT 3 Database Server ebenfalls gestartet und ist über ADS von einem Client-System konfigurierbar.

Einige Datenbanken, wie zum Beispiel die SQLite, können nur benutzt werden, wenn das entsprechende Package installiert wurde.

4 Technische Einführung

4.1 Grundkonzept

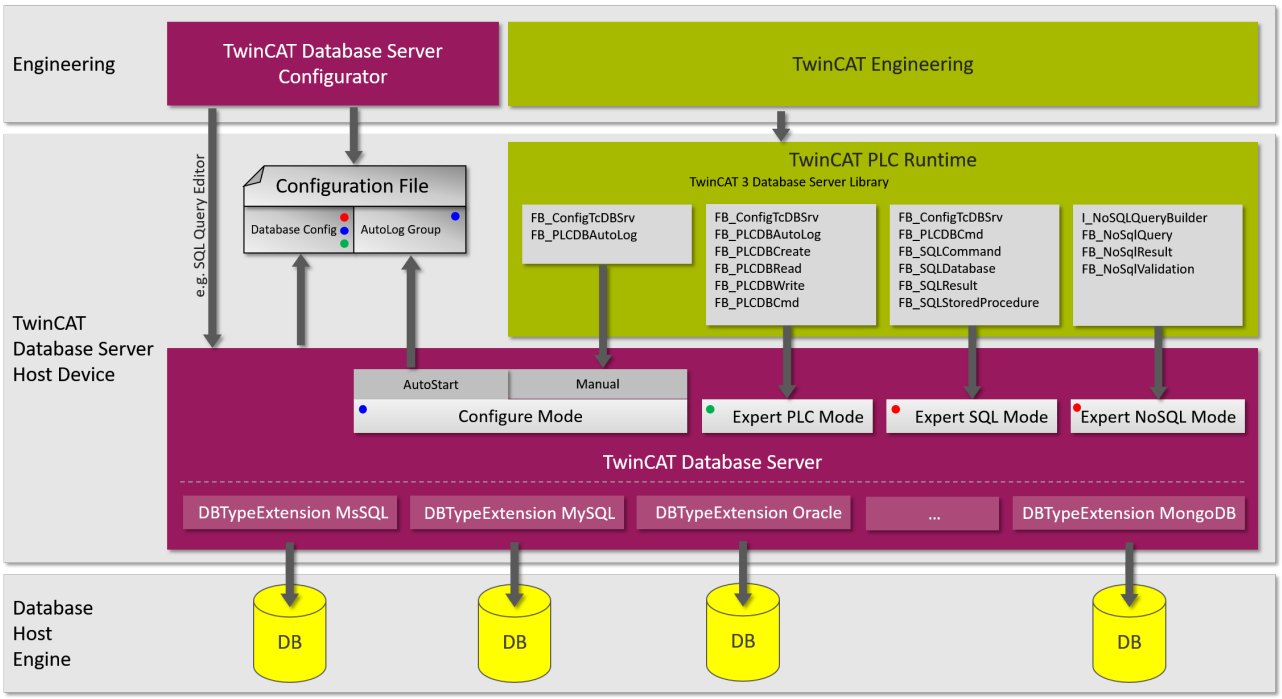
Der TwinCAT Database Server soll es allen TwinCAT-Nutzern ermöglichen, möglichst leicht eine Datenbankanbindung der Steuerung zu realisieren. Da die volle Flexibilität aufgrund der Einfachheit nicht verloren gehen soll, bietet der TwinCAT Database Server in seinem Grundprinzip vier wesentliche Kategorien:

- **Configure Mode:Reine Konfigurationslösung**
  Datenbankanbindungen für einfache Applikationen auf Basis von grafischen Konfigurationen ohne Code-Implementierung.
- **PLC Expert Mode: Programmierlösung für klassische SPS-Programmierer**
  Datenbankanbindung für einfache bis komplexe Applikationen auf Basis von SPS-Funktionsbausteinen, bei denen die Datenbank-Kommandos weitgehend automatisch vom Database Server generiert werden.
- **SQL Expert Mode: Programmierlösung für klassische SPS-Programmierer und Datenbankexperten**
  Datenbankanbindung für einfache bis komplexe Applikationen auf Basis von SPS-Funktionsbausteinen und C++-Interfaces, bei denen die Datenbank-Kommandos im Programmablauf eigenständig zusammengebaut werden. Für höchste Flexibilität.
- **NoSQL Expert Mode: Programmierlösung für SPS-Programmierer und NoSql Datenbankexperten**
  Datenbankanbindung für einfache bis komplexe Applikationen mit SPS-Funktionsbausteinen, bei denen NoSql-Kommandos im Programmablauf erstellt und abgeschickt werden können.

Natürlich können auch alle vier Kategorien in einer Applikation miteinander kombiniert werden.

Der TwinCAT Database Server kann über einen grafischen Konfigurator eingestellt werden. Dabei wird die Konfiguration in eine XML-Datei geschrieben, welche dann auf das entsprechende Zielsystem heruntergeladen werden kann.
Die Konfigurationsdatei liegt auf „nicht Windows CE“-Geräten im Ordner *C:\TwinCAT\3.1\Boot* und auf Windows-CE-Geräten im Ordner *\Hard Disk\TwinCAT\ 3.1\Boot*.
Es ist möglich, lesend und schreibend auf verschiedene Datenbanksysteme zuzugreifen. Dafür stehen abwählbare Datenbankerweiterungen zur Verfügung. Die unterstützten Datenbanken werden im Abschnitt „Datenbanken [▶ 126]“ beschrieben.

Der TwinCAT Database Server selbst ist ein Service, der zusammen mit dem TwinCAT-System auf dem jeweiligen Steuerungsrechner gestartet wird. Er ist das Bindeglied zwischen SPS und Datenbank.



Configure Mode

Im Configure Mode ist der Hauptteil der Arbeit im Konfigurator zu verrichten. Hier muss die Konfiguration für die gewünschte Datenbank und für die AutoLog-Gruppe eingestellt werden. Für die Konfiguration der AutoLog-Gruppe kann der Target Browser genutzt werden, um online auf ein Zielsystem zuzugreifen und die zu kommunizierenden Variablen auszuwählen. Nutzt der Anwender die Option **AutoStart**, so wird die Kommunikation mit der konfigurierten Datenbank direkt mit dem Aufstarten des TwinCAT-Systems aufgenommen. Wird die Option **Manual** gewählt, muss die Kommunikation durch den Funktionsbaustein FB_PLCDBAutoLog [▶ 164] oder für den Autolog View angesteuert werden.

PLC Expert Mode

Im PLC Expert Mode wird nur die Datenbankkonfiguration im Konfigurator eingestellt. Weitere Funktionalitäten werden im SPS-Code der Applikation implementiert. Mit dem Funktionsbaustein FB_PLCDBCreate [▶ 175] ist es sogar möglich, ohne den Konfigurator auszukommen und selbst die Datenbank aus der SPS heraus zu konfigurieren. Ansonsten stehen Funktionsbausteine zum Lesen und Schreiben zur Verfügung. Der Funktionsbaustein FB_PLCDBCmd [▶ 187] bildet dabei den Übergang zwischen PLC Expert Mode und SQL Expert Mode. Hier können Tabellenstrukturen sehr einfach als SPS-Struktur abgebildet und ein SQL-Kommando mit Platzhaltern für die aktuellen Werte der Struktur an den TwinCAT Database Server übergeben werden. Der TwinCAT Database Server setzt dann selbständig alle Werte ein und schickt das Kommando an die Datenbank.

SQL Expert Mode

Im SQL Expert Mode kann sich der Anwender die SQL-Kommandos für zum Beispiel Insert, Select oder Update selber in der SPS zusammenbauen und über den TwinCAT Database Server zur Datenbank schicken. Ein sehr flexibler und performanter Weg. Auch sogenannte Stored Procedures [▶ 205], welche in der Datenbank hinterlegt sind, können aus der SPS aufgerufen werden.

Loggen von Strukturen

Beachten Sie beim Loggen von Strukturen das entsprechende Byte-Alignment.

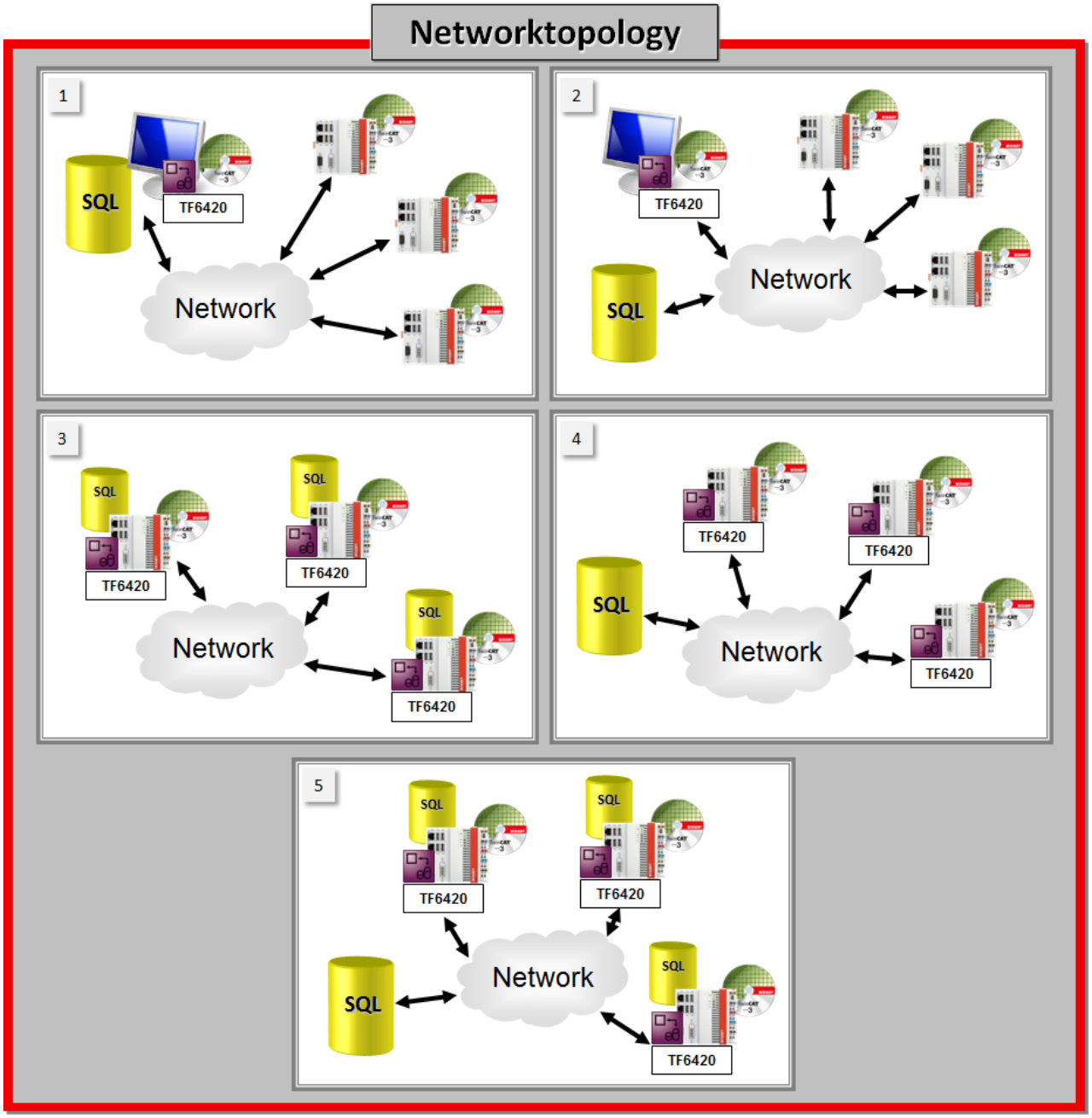
NoSql Expert Mode

Im NoSql Expert Mode kann sich der Anwender NoSql-Abfragen für z. B. Insert, Find und viele weitere datenbankspezifische Abfragen zusammenstellen und über den TwinCAT Database Server zur Datenbank schicken. Neue und flexiblere Datenschemata, wie hierarchische Strukturen und Arrays, werden dabei unterstützt.

4.2 Einsatzgebiete und Netzwerktopologien

Der TwinCAT Database Server kann in jeder steuerungstechnischen Applikation eingesetzt werden: in Produktionsmaschinen zum Auslesen von Rezepturdaten, im Produktbereich zum Schreiben von Produktionsdaten, beim Condition Monitoring oder bei der Maschinenführung, bei Windkraftanlagen, bei denen Betriebszustände mitgeschrieben werden oder in der Gebäudetechnik. Der Database Server kann in die jeweils vorhandene Netzwerkarchitektur integriert werden.

Die Netzwerktopologie wird meist durch den Datenbanktyp, die örtlichen Begebenheiten und den Anwendungsbereich beeinflusst. Im folgenden Schaubild sind verschiedene Netzwerktopologien aufgezeigt, in denen der TwinCAT Database Server eingesetzt werden kann.



1. TwinCAT und der TwinCAT Database Server sind gemeinsam mit der Datenbank auf einem Rechner. Der Database Server kann über ADS als Gateway für viele Steuerungen dienen. Die Performance muss beachtet werden.
2. TwinCAT und der TwinCAT Database Server sind gemeinsam auf einem Rechner und die Datenbank ist auf einem externen Gerät. Auch hier kann der Database Server über ADS als Gateway für viele Steuerungen dienen. Die Performance muss beachtet werden.
3. Der TwinCAT Database Server ist lokal auf jedem Steuerungsgerät, auf dem auch jeweils eine Datenbank installiert ist. Nicht jede Datenbank ist für einen solchen Einsatz geeignet.
4. Der häufigste Anwendungsfall. Der TwinCAT Database Server ist auf jedem Steuerungsgerät installiert und die Datenbank befindet sich auf einem externen Server im Netzwerk.
5. Kombination aus Fall 3 und Fall 4. Eine Hauptdatenbank liegt im Netzwerk auf einem Server und die Steuerungen im Feld haben jeweils eine lokale Datenbank, welche z.B. bei einem Verbindungsabbruch einspringt und die Daten zunächst lokal speichert. Der Database Server ist auf jedem Steuerungsgerät installiert.

Remote-Zugriff des TwinCAT Database Servers auf eine Datenbank

Für den reibungslosen remote Zugriff des TwinCAT Database Servers auf eine Datenbank müssen Sie auf Datenbankseite verschiedene Aspekte beachten:

- Ist der Remote-Zugriff generell erlaubt?
- Wie viele gleichzeitige Verbindungen sind erlaubt? (Falls der TwinCAT Database Server mehrere Verbindungen öffnen muss)
- Hat der Benutzer, mit dem sich der Database Server an der Datenbank anmelden soll, genügend Rechte?
- Ist die Firewall des Remote-Systems entsprechend konfiguriert?

Genauere Hinweise zur Konfiguration Ihres Datenbankservers können Sie der zugehörigen Datenbank-Dokumentation [▶ 126] entnehmen.

4.3 Kompatibilität

Der TwinCAT Database Server ist seit vielen Jahren ein bewährtes TwinCAT-Produkt. Die Anforderungen an das Produkt steigen stetig. Neue Entwicklungen im TwinCAT Database Servers sollen diesen gestiegenen Anforderungen gerecht werden.

Bisher sind die TwinCAT-Datenbankanbindungen in den Versionen 3.**0**.x, 3.**1**.x und 3.**2**.x im Einsatz. Die aktuelle Version trägt die Versionsnummer 3.**3**.x Wie bisher besteht der Database Server aus den Komponenten Konfigurator, ADS-Server und SPS-Bibliothek. Die Version 3.0.x beinhaltet dabei die SPS-Bibliothek Tc2_Database.compiled-library. Die SPS-Bibliothek in den Versionen 3.1.x und größer heißt Tc3_Database.compiled-library.

Übersicht über freigegebene Versionen des Database Server

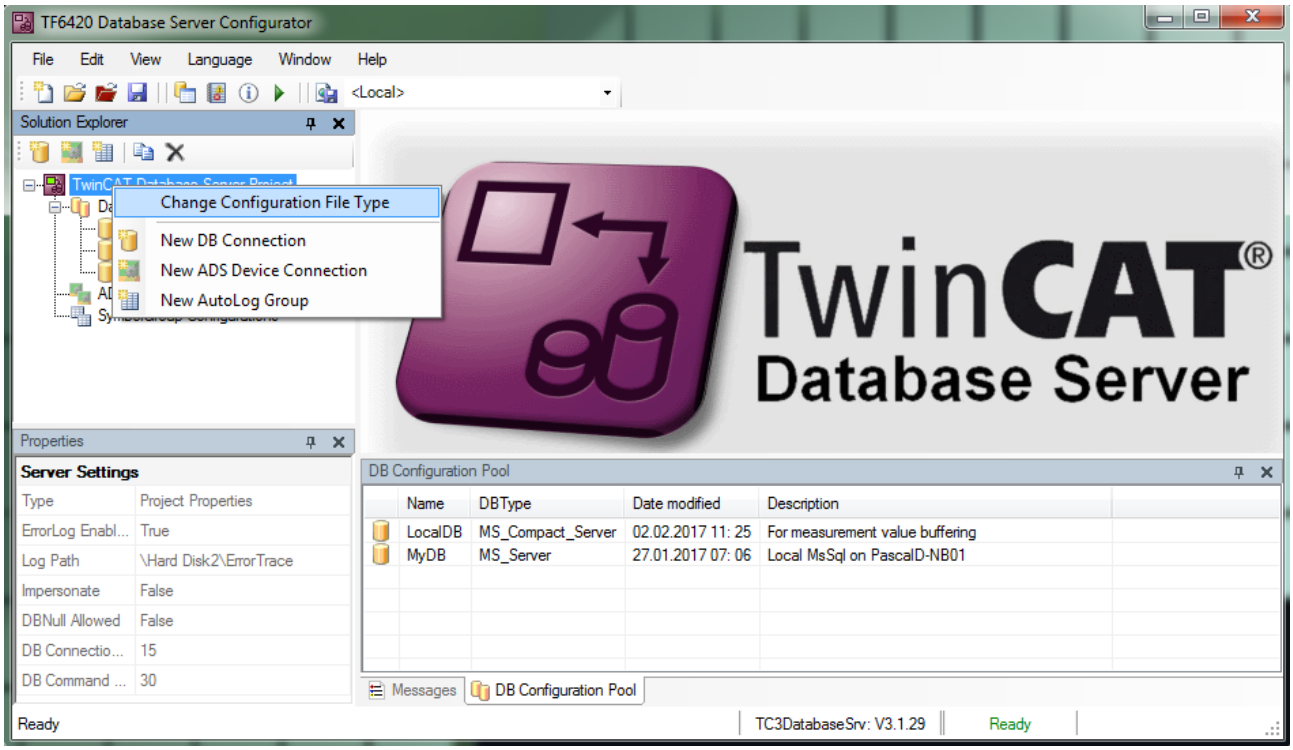
Database Server 3.0.x	3.0.23	3.0.26	3.0.27	3.0.28					
Database Server 3.1.x					3.1.29	3.1.30	3.1.31		
Database Server 3.2.x								3.2.32	
Database Server 3.3.x									3.3.33

Hinweise zum Übergang von 3.0.x zu 3.1.x

Neben neuen und performanteren Funktionen, lag ein Hauptaugenmerk auf der Kompatibilität zwischen Version 3.0.x und 3.1.x. So kann beispielweise alter SPS-Code, in dem die Tc2_Database.compiled-library verwendet wird, auch mit dem neuen ADS-Server der Version 3.1.x verwendet werden. Die alte Tc2_Database.compiled-library wird auch mit der Version 3.1.x weiterhin durch das Setup installiert. Die durch den Konfigurator für den Server erzeugten XML-Dateien unterscheiden sich zwischen Version 3.0.x und 3.1.x. Jedoch ist es möglich, auch alte Konfigurationsdateien mit dem neuen Konfigurator (Standalone) einzulesen und diese sogar bei Bedarf in das neue Format zu konvertieren.

Sicherung der alten XML-Konfiguration

Beim Updaten des TwinCAT Database Servers 3.0.x auf die neue Version 3.1.x, wird durch das Setup die alte XML-Konfiguration gesichert. Sie wird umbenannt in „CurrentConfigDataBase**_OLD**.xml“ und liegt weiterhin im TwinCAT-Boot-Verzeichnis.



Letztendlich kann ein alter Konfigurator und ein alter ADS-Server der Versionen 3.0.x nicht mit der neuen Tc3_Database.compiled-library verwendet werden, da diese Komponenten zuvor definiert wurden. Eine Übersicht gibt die nachfolgende Grafik.

	Konfigurator 3.0.x	Server 3.0.x	Tc2_Database.compiled-library	Konfigurator 3.1.x Standalone	Konfigurator 3.1.x Visual Studio	Server 3.1.x	Tc3_Database.compiled-library
Konfigurator 3.1.x Standalone		✓	✓			✓	✓
Konfigurator 3.1.x Visual Studio		✗	✓			✓	✓
Server 3.1.x	✓		✓		✓		✓
Tc3_Database.compiled-library	✗	✗			✓	✓	

Hinweise zum Übergang von 3.1.x zu 3.2.x

Die Dateiformate für die Konfigurationen sind alle unverändert. Der ADS-Server wurde nur um neue Funktionalitäten erweitert. Alle anderen Funktionen haben immer noch Bestand. Die alte Tc2_Database.compiled-library wird neben der Tc3_Database.compiled-library weiter mit der Version 3.2.x durch das Setup installiert. Es gelten die Hinweise für den Übergang von 3.0.x zu 3.1.x.

In der Tc3_Database.compiled-library sind ab der Version 3.2.x alle bisherigen Funktionsbausteine aktualisiert worden. Die Aktualisierung bezieht sich auf das Eventlogger Interface I_TcMessage. Damit alte Applikationen weiterhin funktionieren, sind neue Bausteine im Namen am Ende mit „Evt“ gekennzeichnet. Alle alten Bausteine sind immer noch in der Bibliothek enthalten, sind jedoch im Ordner Obsolete und werden auch von dem Kompiler entsprechend gekennzeichnet.

Beispiel:

In Version 3.1.x: FB_SQLCommand

In Version 3.2.x: FB_SQLCommandEvt

Es wird empfohlen bei neuen Projekten die Bausteine mit der Endung „Evt“ zu nutzen. Es ist dabei zu beachten, dass der Eventlogger selbst, erst ab dem TwinCAT 3.1 Build 4022.20 verfügbar ist und die Bausteine somit erst ab 4022.20 nutzbar sind.

Hinweise zum Übergang von 3.2.x zu 3.3.x

Mit der Version 3.3.x wurde die Modularisierung des TwinCAT Database Servers auf Treiber-Seite weiter vorangetrieben. Es besteht volle Kompatibilität. Manche Treiber sind allerdings nicht mehr im Setup automatisch enthalten und müssen von dem Nutzer selbst Verfügbar gemacht werden. Beispiel: MySQL unter Windows CE. Details dazu entnehmen Sie bitte im Bereich Konfiguration den Einstellungen für die jeweilige Datenbank.

5 Konfiguration

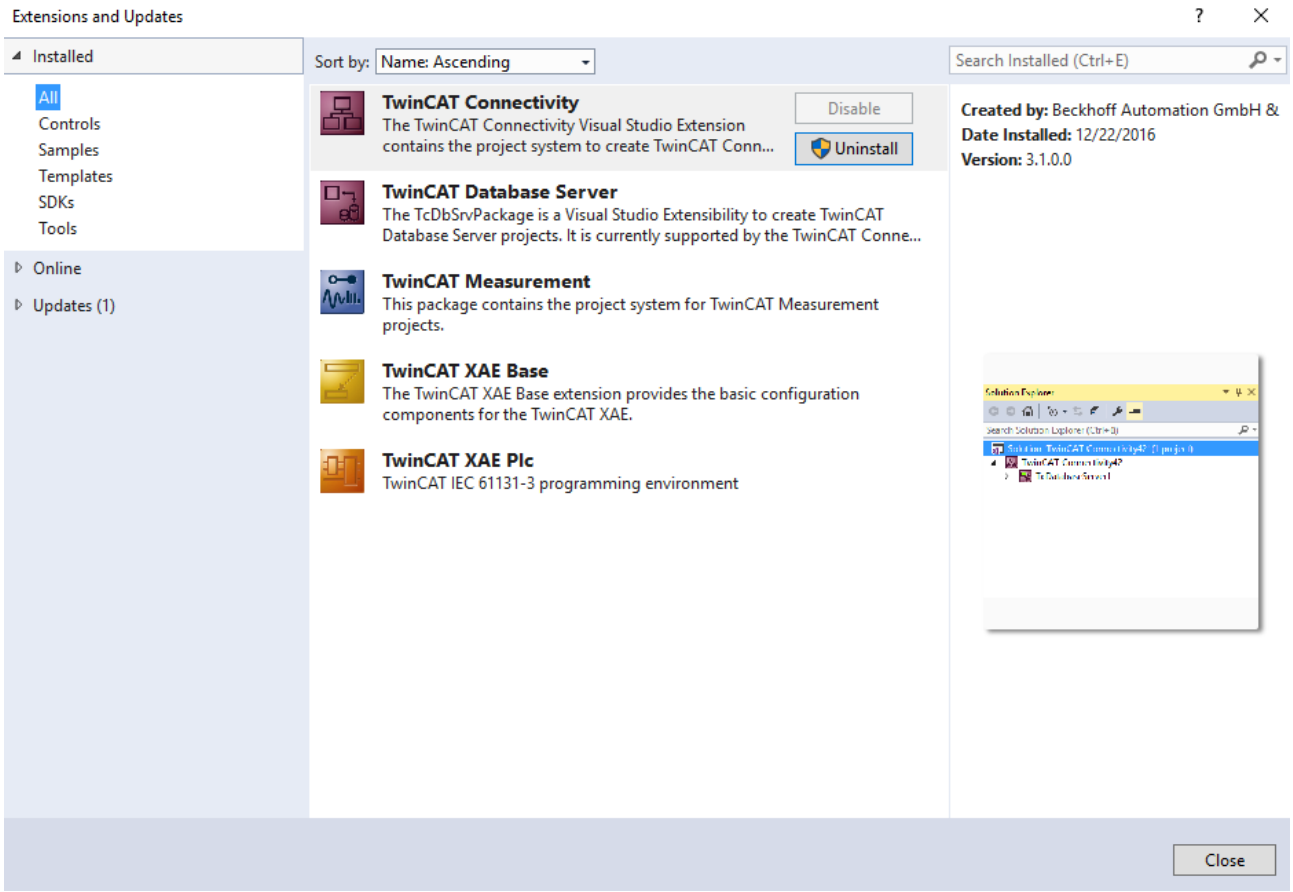
5.1 Konfigurator

Der TwinCAT Database Server wird über den Konfigurator eingestellt und gesteuert. Außerdem bietet das Tool zahlreiche Entwicklungshilfen, um die Entwicklung der Applikation in der SPS zu beschleunigen.

Um die Entwicklung möglichst komfortabel zu gestalten, ist der Konfigurator in Visual Studio integriert. TwinCAT-Projekte und TwinCAT-Database-Server-Projekte können in einer gemeinsamen Solution platziert werden. Zusätzlich kann der Kunde auch weiterhin unabhängig vom Visual Studio den Standalone-Konfigurator verwenden.

5.1.1 Integration in Visual Studio

Der TwinCAT Database Server ist in Visual Studio 2013 und Visual Studio 2015 integriert. Diese Integration wird mithilfe von zwei Erweiterungen erreicht. Im Erweiterungsfenster von Visual Studio werden bei der Installation zwei notwendige Erweiterungen hinzugefügt, welche unter anderem die Funktionalitäten des Projektsystems beinhalten.

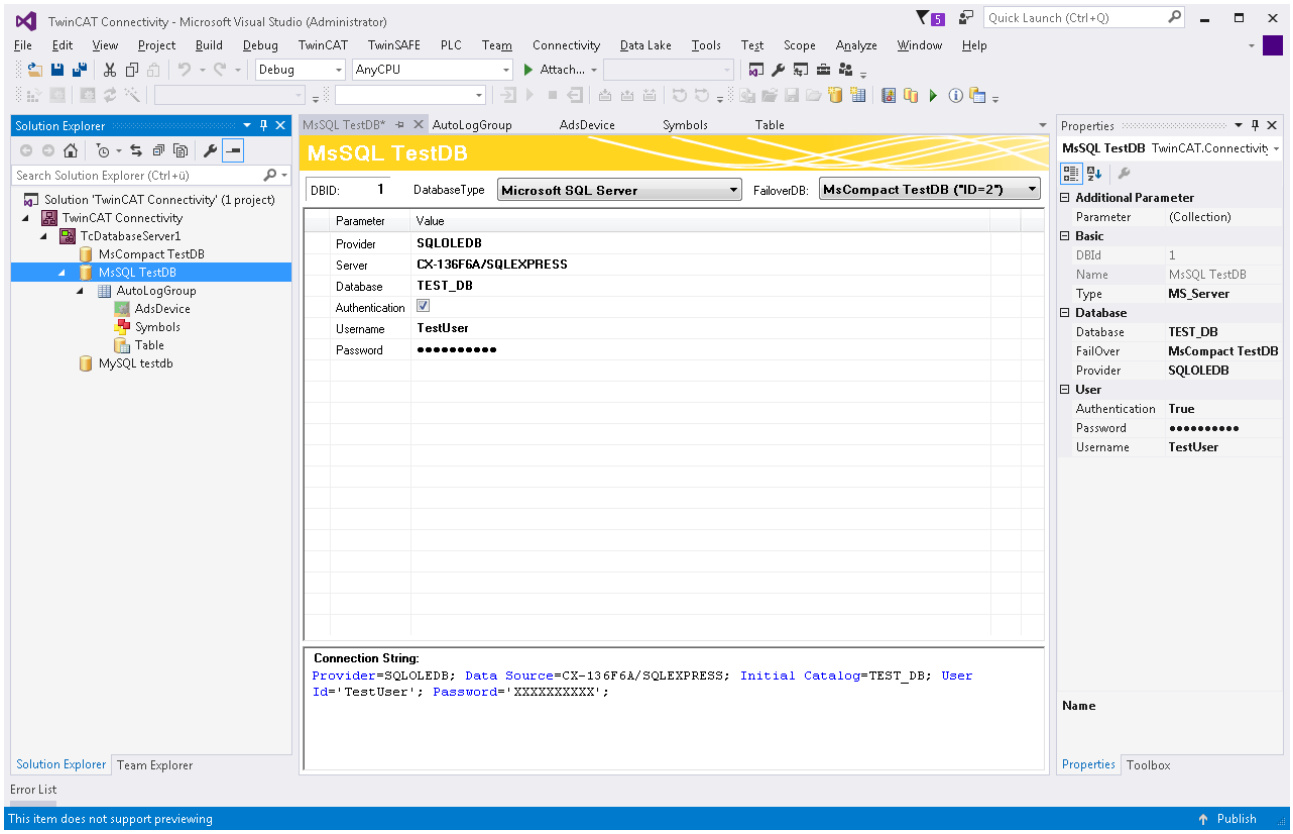


5.1.1.1 Allgemein

In diesem Kapitel werden die einzelnen Funktionen und Komponenten der Visual Studio Integration vom TwinCAT Database Servers erklärt.

5.1.1.1.1 Komponenten der Benutzeroberfläche

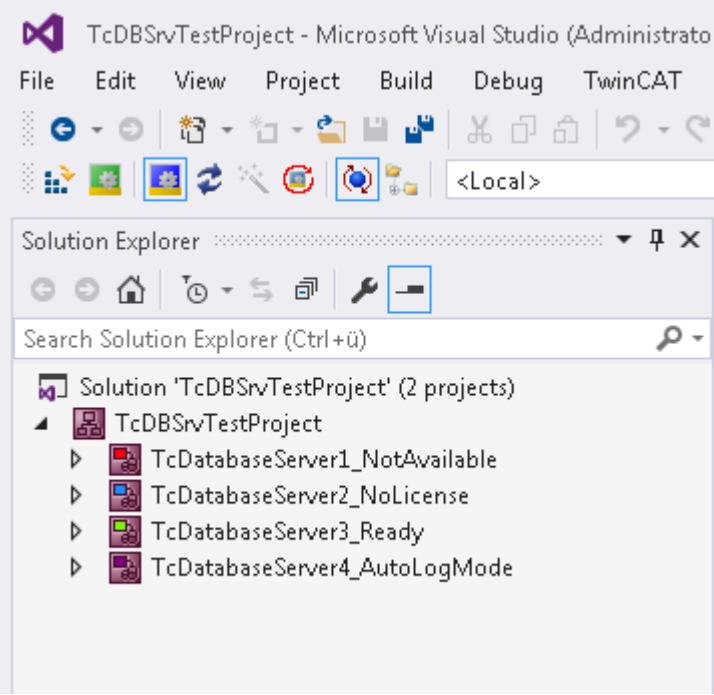
Der TwinCAT Database Server ist in Visual Studio 2013 und Visual Studio 2015 integriert. Dafür verfügt Visual Studio mit der TwinCAT-Connectivity-Erweiterung über ein neues Projektsystem. In diesem kann unter anderem ein dateibasierendes TwinCAT-Database-Server-Projekt angelegt werden. Die typischen Bestandteile, wie das Eigenschaftsfenster, der Solution Explorer oder die Fehlerausgabe werden unterstützt. Außerdem werden diverse Editoren zur Bearbeitung der Konfigurationsdateien bereitgestellt.



In ein TwinCAT-Connectivity-Projekt können beliebig viele TwinCAT-Database-Server-Projekte eingebunden werden.

Das Projekt-Icon zeigt dabei den Zustand des eingestellten Zielsystems an:

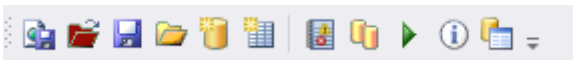
- Rot: Der TwinCAT Database Server ist nicht erreichbar.
- Blau: Der TwinCAT Database Server hat keine gültige Lizenz. (Siehe Lizenzierung [▶ 13])
- Grün: Der TwinCAT Database Server ist erreichbar und einsatzbereit.
- Violett: Der TwinCAT Database Server befindet sich im AutoLog-Modus. Daten zwischen SPS und Datenbank werden ausgetauscht.



Die TwinCAT-Database-Projekte bilden dabei ein dateibasiertes Projektsystem ab. Im Solution Explorer werden die einzelnen Konfigurationsdokumente verwaltet. Wenn ausstehende Änderungen in den Editoren vorliegen, werden die Dokumente durch ein * markiert. Werden Änderungen ohne Öffnen des Dokumentes vorgenommen (durch das Eigenschaftenfenster), werden die Änderungen trotzdem registriert. Im Abschnitt „TwinCAT Database Server Projekt [▶ 28]“ finden Sie weitere Informationen zum Projektsystem.

Toolbar und Kommandos

Die Toolbar besteht aus folgenden Elementen:

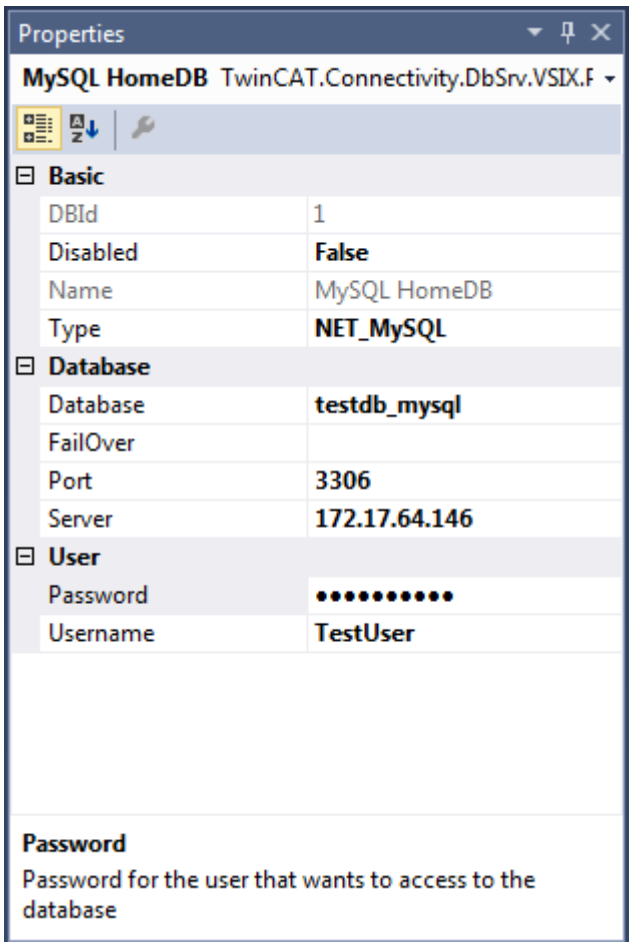


Toolstrip Button	Beschreibung
	Aktivieren der Konfiguration
	Konfiguration vom Zielgerät lesen
	Konfiguration in eine XML-Datei abspeichern
	Konfiguration aus einer XML-Datei einlesen
	Neue Datenbankkonfiguration hinzufügen
	Neue AutoLog-Gruppe hinzufügen
	Eventanzeige
	Datenbankpool
	AutoLog Viewer
	Information LogView
	SQL Query Editor

Eigenschaftenfenster

Die Einstellungen der unterschiedlichen Projektdokumente können über dafür vorgesehene Editoren oder über das Eigenschaftenfenster konfiguriert werden. Dabei werden die Dateiinhalte verändert, nicht die Metadaten in der Projektdatei des TwinCAT-Connectivity-Projektes.

Zu den einzelnen Eigenschaften befindet sich im unteren Bereich des Eigenschaftenfensters eine detailliertere Beschreibung. Beachten Sie, dass einige Listen nur im Editor bearbeitet werden können.



Ausgabe und Fehlerliste

Das Visual Studio verfügt über eine integrierte Konsolenausgabe. Der TwinCAT Database Server nutzt diese Ausgabe, um aktuelle Benachrichtigungen, Warnungen oder Fehler auszugeben. Dazu kann in der Ausgabe die Kategorie „TwinCAT Database Server“ ausgewählt werden. Es ist möglich, dass diese Kategorie noch nicht vorhanden ist, da noch keine Meldung vom TwinCAT Database Server aufgetreten ist.

Zusätzlich wird die Fehlerliste von Visual Studio dazu genutzt, um auch hier die wichtigsten Informationen mitzuteilen.

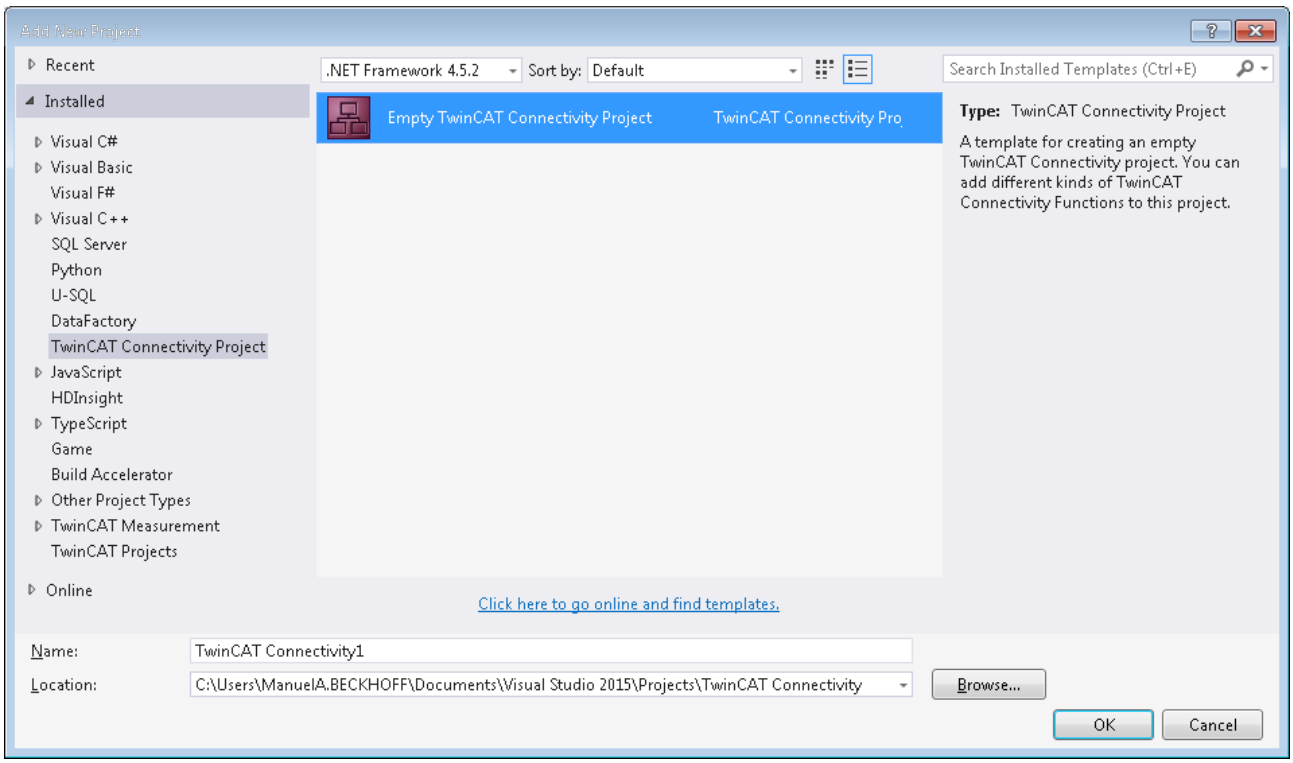
Beide Fenster können Sie im Visual Studio über **Ansicht > Fehlerliste/Ausgabe** öffnen.

5.1.1.1.2 TwinCAT-Database-Server-Projekt

Projekt erstellen

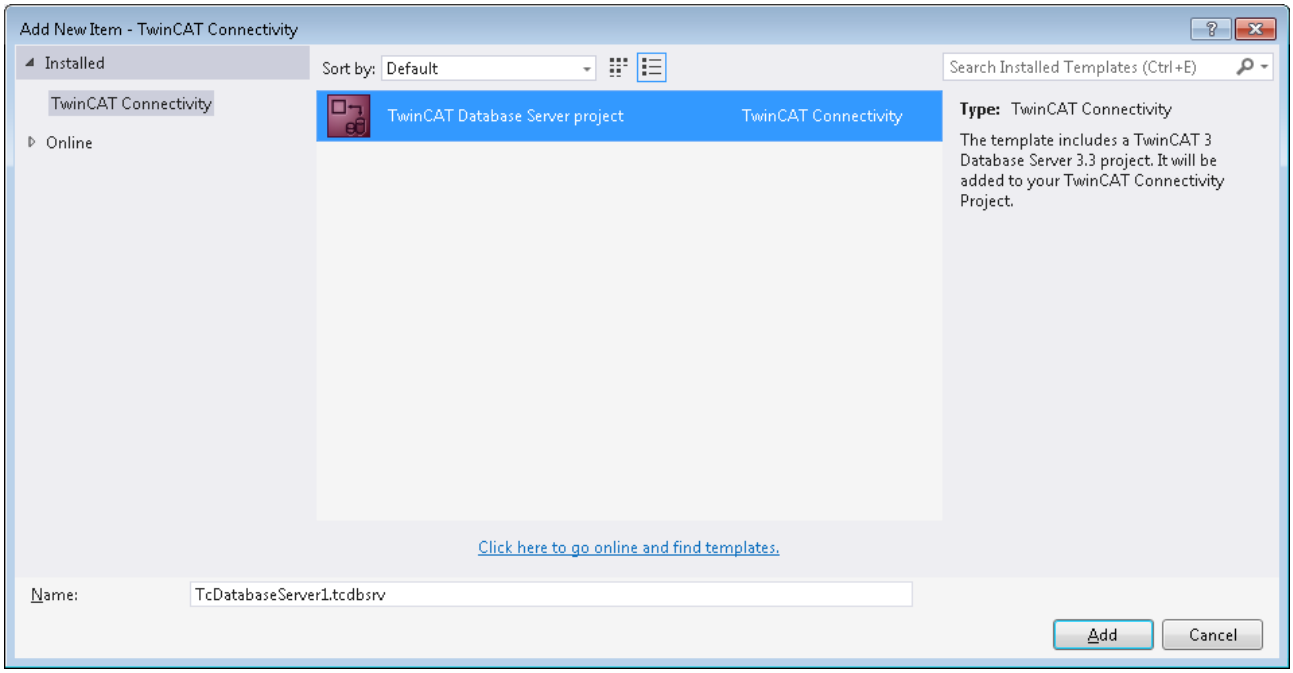
Durch die TwinCAT-Connectivity-Erweiterung für Visual Studio steht eine neue Projektvorlage zur Verfügung. Beim Erstellen eines neuen Projektes erscheint nun die Kategorie **TwinCAT Connectivity Project** in der Auswahl.

Um ein neues TwinCAT-Connectivity-Projekt zu erstellen, wählen Sie das **Empty TwinCAT Connectivity Project**, legen den Projektnamen und den Speicherort fest und fügen es mit **OK** der Solution hinzu. TwinCAT-Connectivity-Projekte bzw. TwinCAT-Database-Server-Projekte können so komfortabel neben TwinCAT- oder anderen Visual-Studio-Projekten angelegt werden.

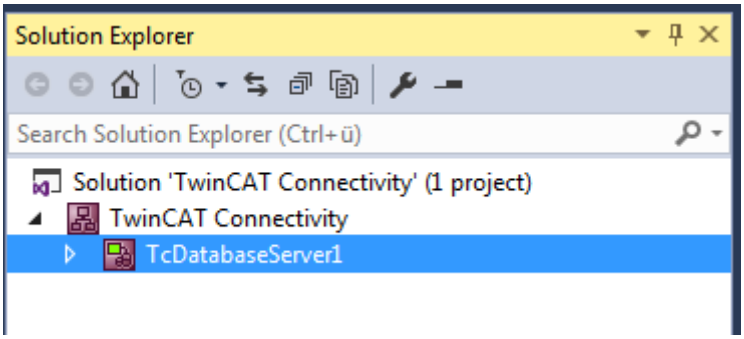


In der Solution erscheint ein neuer Projektknoten. Unterhalb des Connectivity-Projektknotens können Sie Subprojekte der unterstützten Connectivity-Funktionen ergänzen.

Mit **Add** können Sie dem TwinCAT-Connectivity-Projekt ein neues TwinCAT-Database-Server-Projekt hinzufügen. Das TwinCAT-Database-Server-Projekt befindet sich in der Auflistung der vorhandenen Item Templates.



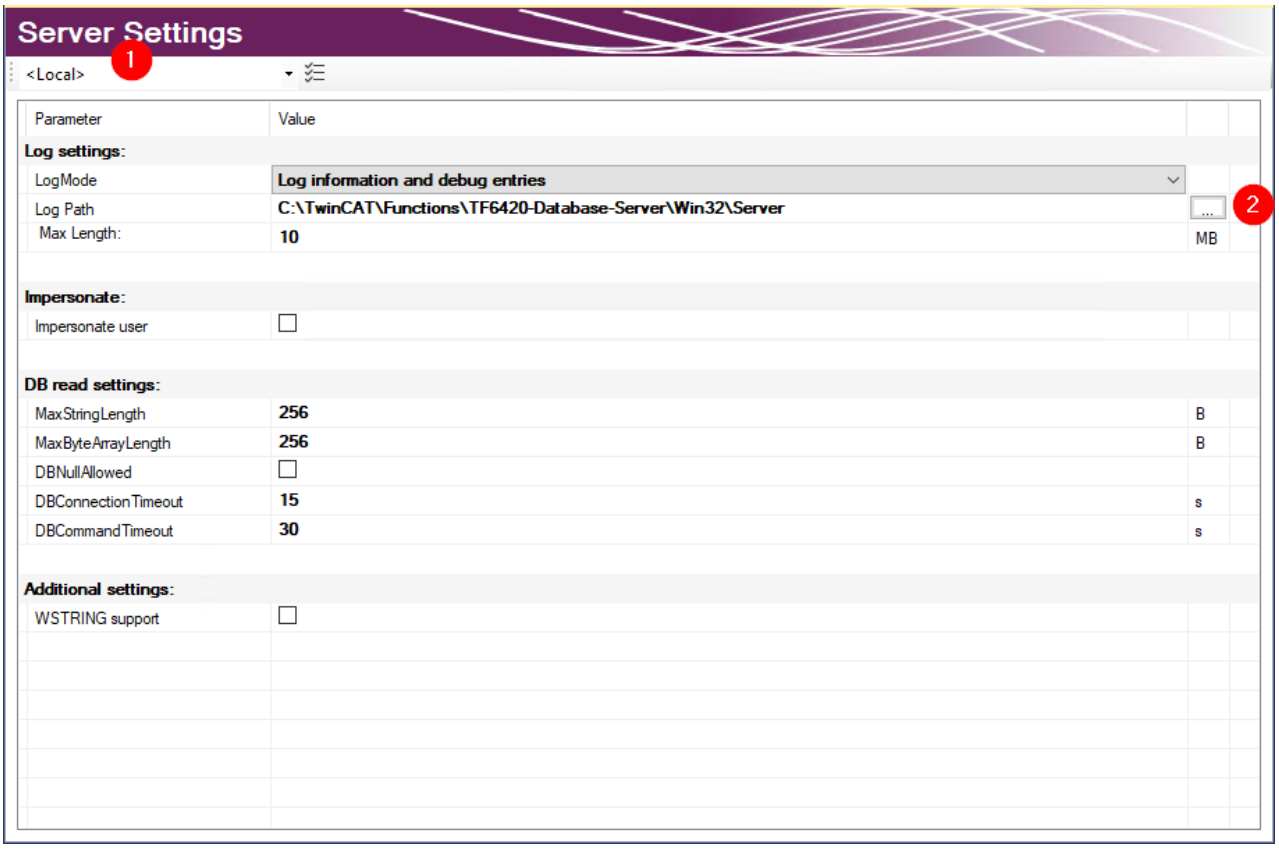
Unterhalb des TwinCAT-Connectivity-Knotens wird ein neues TwinCAT-Database-Server-Projekt angelegt.



Dieses dient nun als Basis für die anstehende Konfiguration eines TwinCAT Database Servers. Das Dokument können Sie sowohl über die Eigenschaften im Eigenschaftsfenster, als auch über einen Editor bearbeiten.

Einem Connectivity-Projekt können Sie beliebig viele TwinCAT-Database-Server-Projekte oder andere Projekte hinzufügen, und damit auch mehrere Konfigurationen in einem Connectivity-Projekt einstellen.

Editor für Server-Einstellungen



Im Editor **Server Settings** können Sie die Einstellungen für den TwinCAT Database Server bearbeiten. Diese Einstellungen betreffen den entsprechenden Server im Allgemeinen. Im Drop-down-Menü (1) können Sie das Zielsystem über die Ams-NetId angegeben. Dafür müssen Sie über TwinCAT eine Route zum Zielsystem anlegen. Wird eine fertige Konfiguration übertragen, werden die Einstellungen auf den TwinCAT Database Server dieses Zielsystems gespeichert.

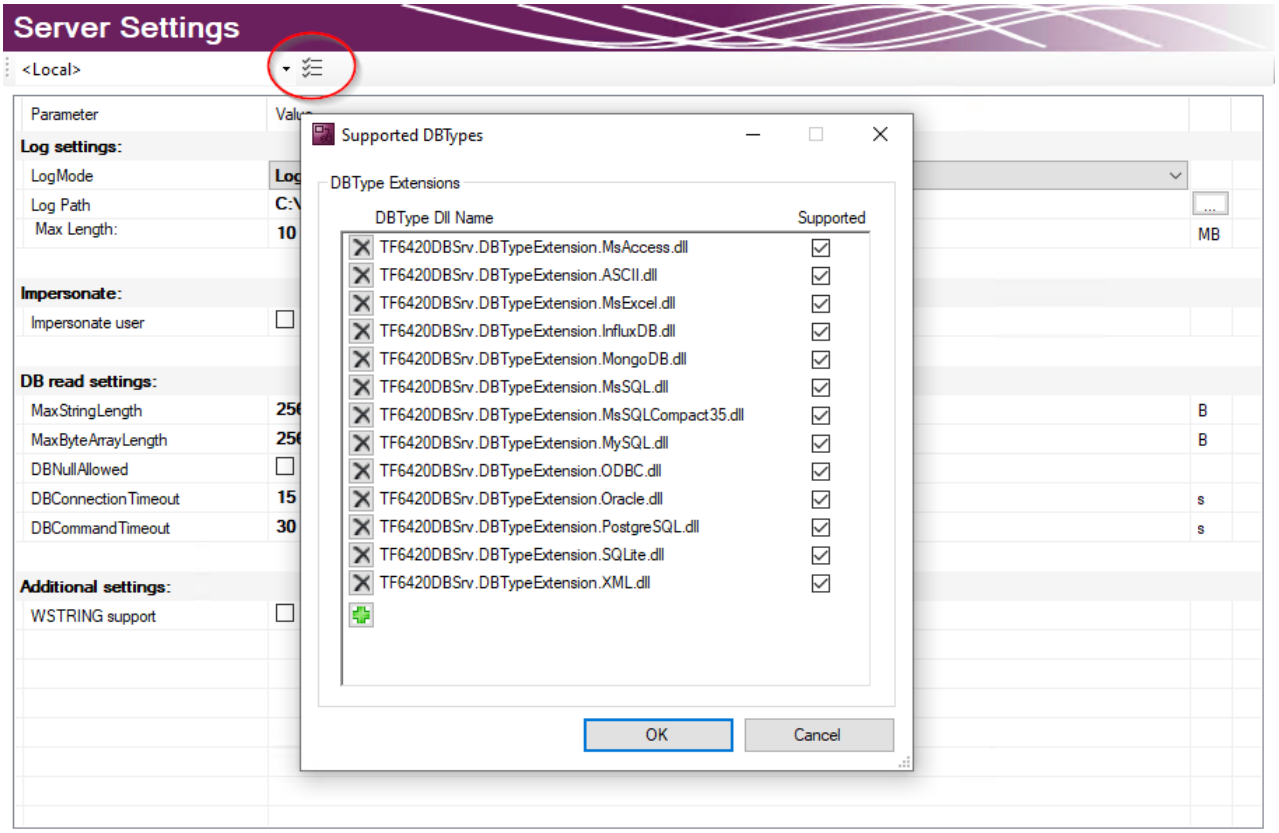
In den **Log settings** konfigurieren Sie Einstellungen zum Aufzeichnen von Fehlerfällen. In einem Fehlerfall erzeugt der Database Server einen detaillierten Eintrag in einer Logdatei. Diese können Sie mit dem Information Log Viewer [▶ 51] auslesen. In den **Log Settings** geben Sie einen Pfad zum Ablageort und die maximale Größe der Datei an. Zusätzlich können Sie die Genauigkeit des Logs beeinflussen. Wir empfehlen, aus Performancegründen, das Loggen nach erfolgter Fehleranalyse wieder abzuschalten, wenn es nicht mehr benötigt wird.

Bei Netzwerkzugriff auf dateibasierende Datenbanken wie Access Datenbanken oder SQL Compact Datenbanken müssen Sie die Option **Impersonate** setzen, damit sich der TwinCAT Database Server mit dem Netzwerklaufwerk verbinden kann. **Diese Funktion wird zurzeit nicht unter Windows CE unterstützt.**

Sie können weitere Einstellungen konfigurieren, um das Lesen aus der Datenbank zu steuern. Diese Einstellungen beziehen sich auf den TwinCAT Database Server auf dem Zielsystem:

MaxStringLength	Maximale String-Länge der Variablen in der SPS
MaxByteArrayLength	Maximale Byte-Array-Länge der Variablen in der SPS
DBNullAllowed	Gibt an, ob NULL-Werte im TwinCAT Database Server angenommen werden.
DBConnectionTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem Verbindungsaufbau von einem Verbindungsfehler ausgeht.
DBCommandTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem abgesendeten Kommando von einem Verbindungsfehler ausgeht. Bei hohen Datenmengen kann, je nach Datenbank und Infrastruktur ein Befehl durchaus sehr viel Zeit in Anspruch nehmen.

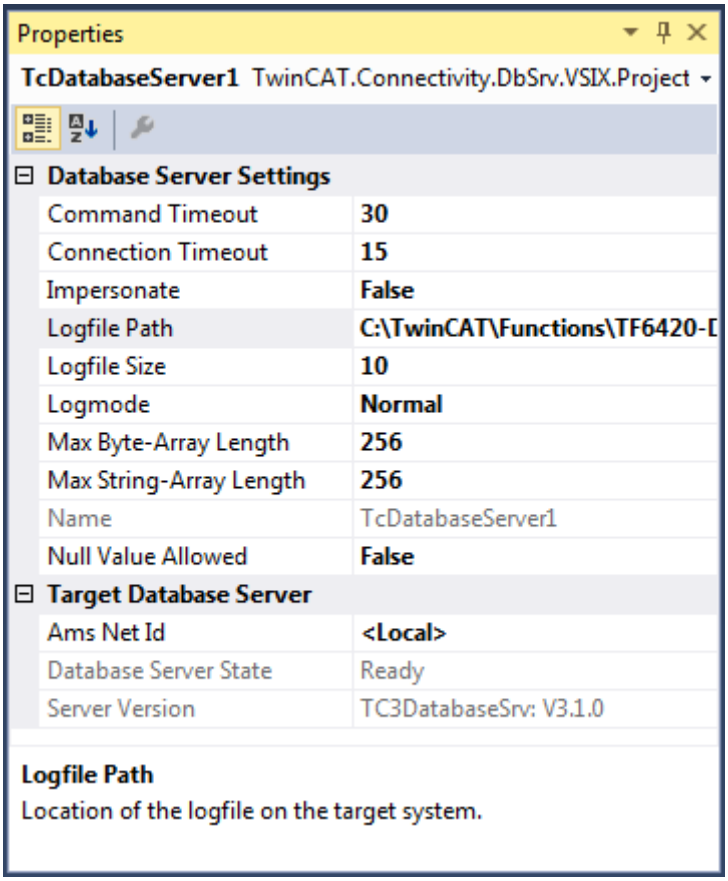
Unterstützte Datenbanktypen



Die installierten Datenbanktypen können in den Server Einstellungen ausgewählt werden. Standardmäßig sind alle installierten Datenbanken angewählt. Der TwinCAT 3 Database Server wird die entsprechenden Datenbankschnittstellen laden. Auf diese Weise können nicht genutzte Datenbanken auf dem Zielsystem abgewählt werden.

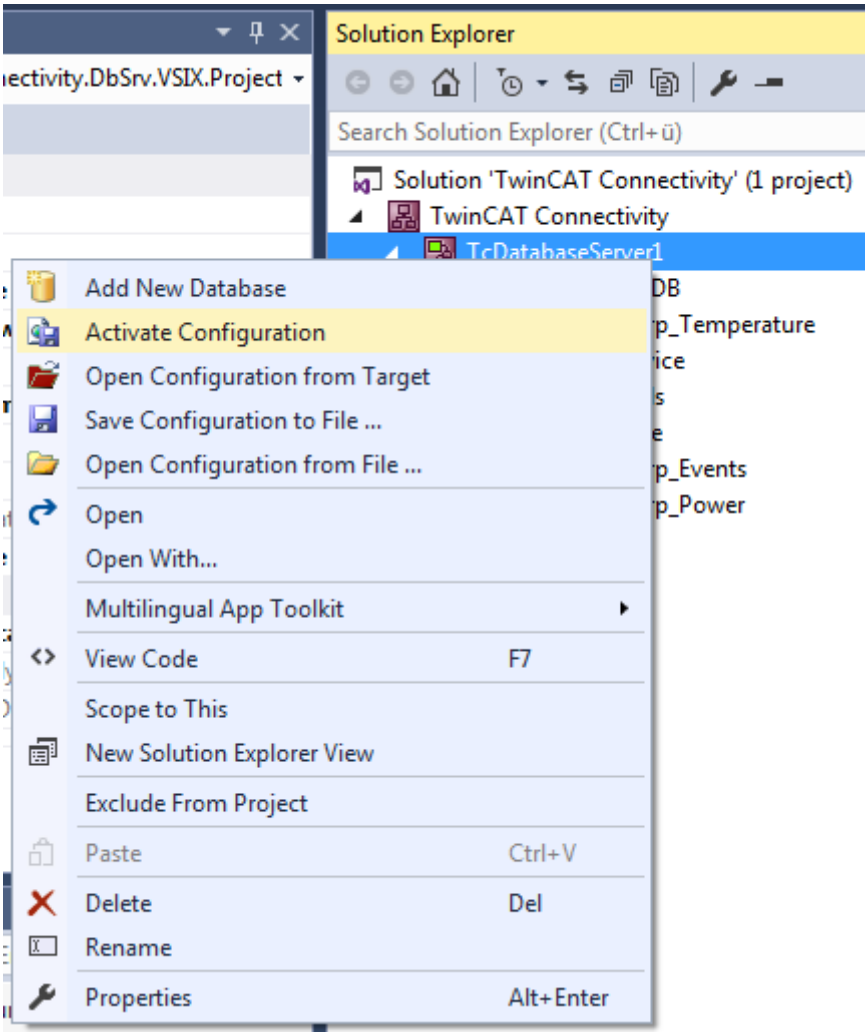
Server-Einstellungen im Eigenschaftenfenster

Die Einstellungen für den TwinCAT Database Server können Sie nicht nur im Editorfenster sondern auch im Eigenschaftenfenster des Database-Server-Projekts vornehmen. Auch diese Eigenschaften wirken sich direkt auf die Konfigurationsdatei aus.



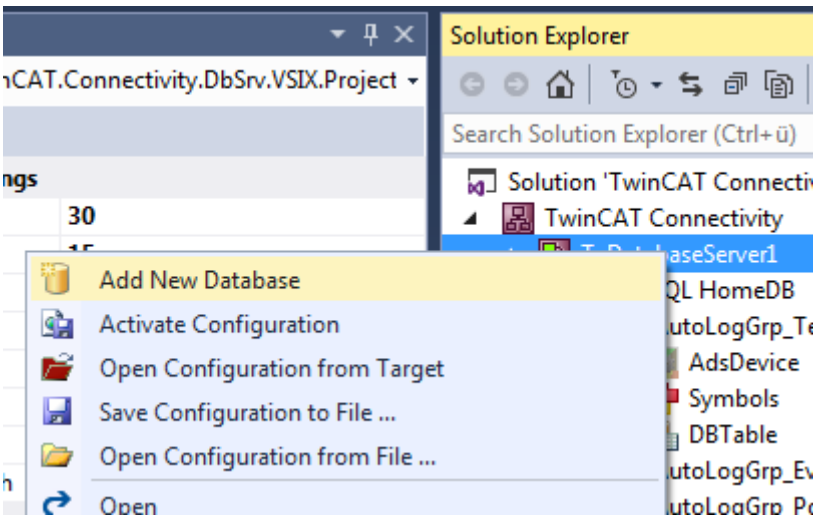
Projekt aktivieren

Um ein konfiguriertes Projekt auf dem TwinCAT Database Server zu aktivieren, verwenden Sie im Kontextmenü des TwinCAT-Database-Server-Projektes das Kommando **Activate Configuration**.



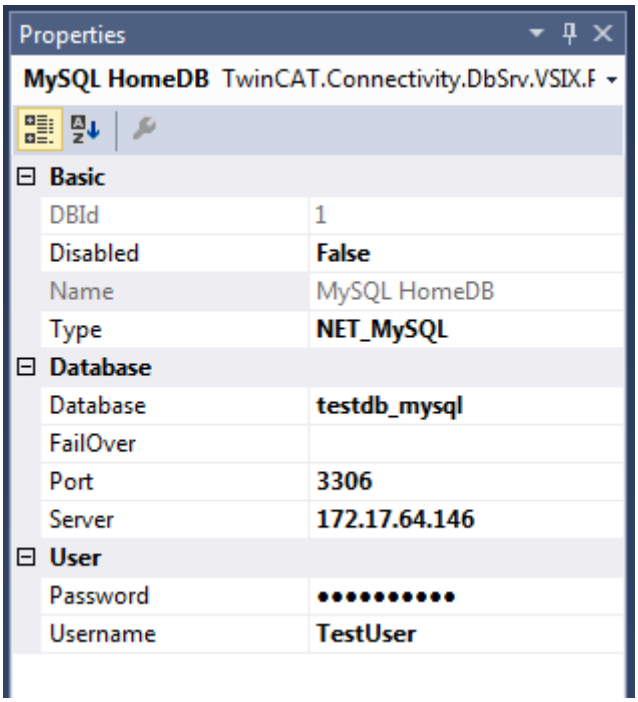
5.1.1.1.3 Datenbanken konfigurieren

Eine neue Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new Database** über das Kontextmenü eines Database-Server-Projektes oder das entsprechende Kommando in der Toolbar hinzufügen.



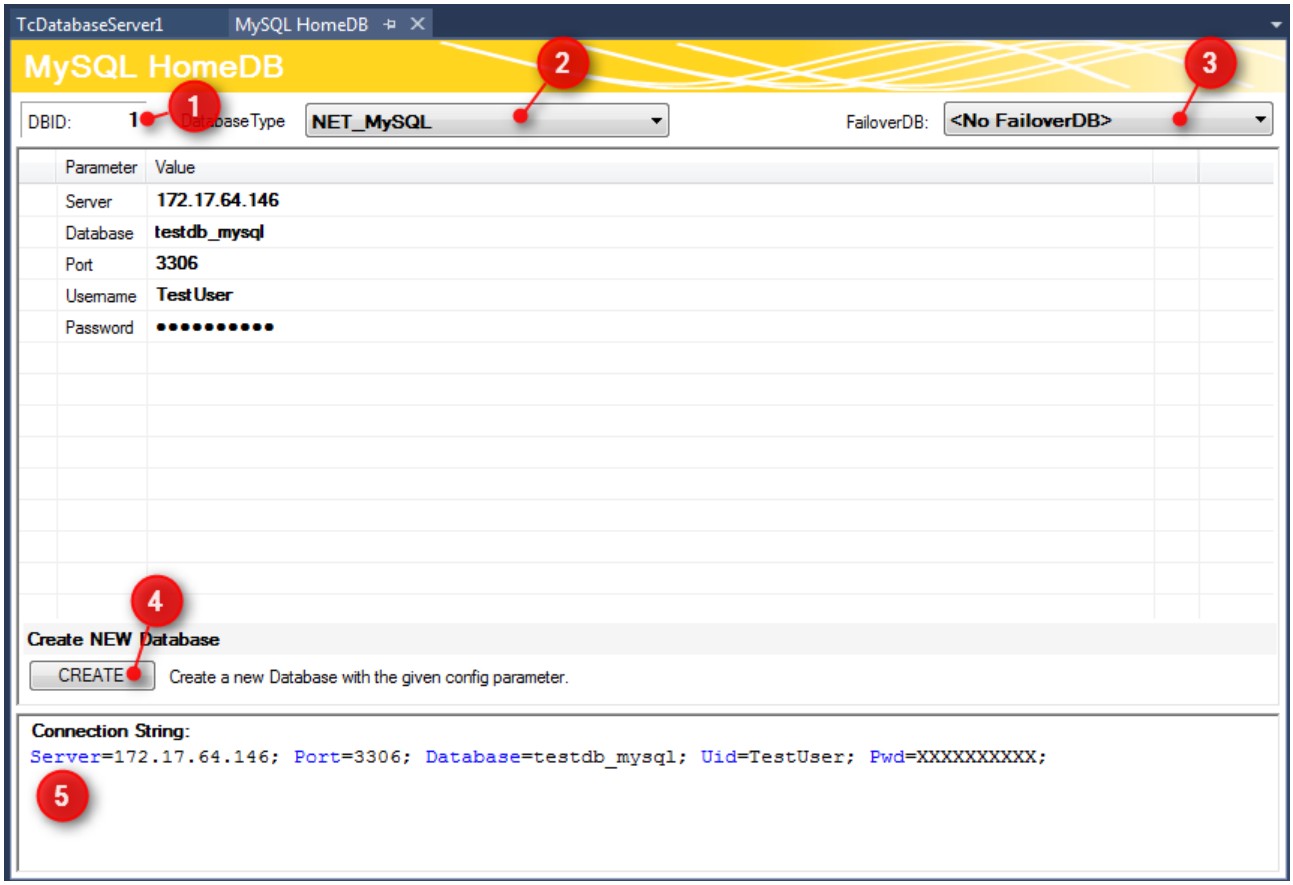
Eine neue Datenbankkonfiguration wird als Datei auf im Projektordner hinzugefügt und in das Projekt eingebunden. Wie bei allen Visual-Studio-Projekten, werden die Informationen über die neuen Dateien im Connectivity-Projekt hinterlegt.

Die neue Datenbankkonfiguration im TwinCAT-Database-Server-Projekt bearbeiten Sie mithilfe des Eigenschaftsfensters oder einem speziellen Editor:



Die Eigenschaften passen sich dabei dynamisch an den jeweils ausgewählten Datenbanktypen an, da die Datenbanken unterschiedliche Parameter haben. Diese Einstellungen betreffen den Inhalt der Datei, nicht die Eigenschaften der Datei selbst.

Editor für Datenbankkonfigurationen

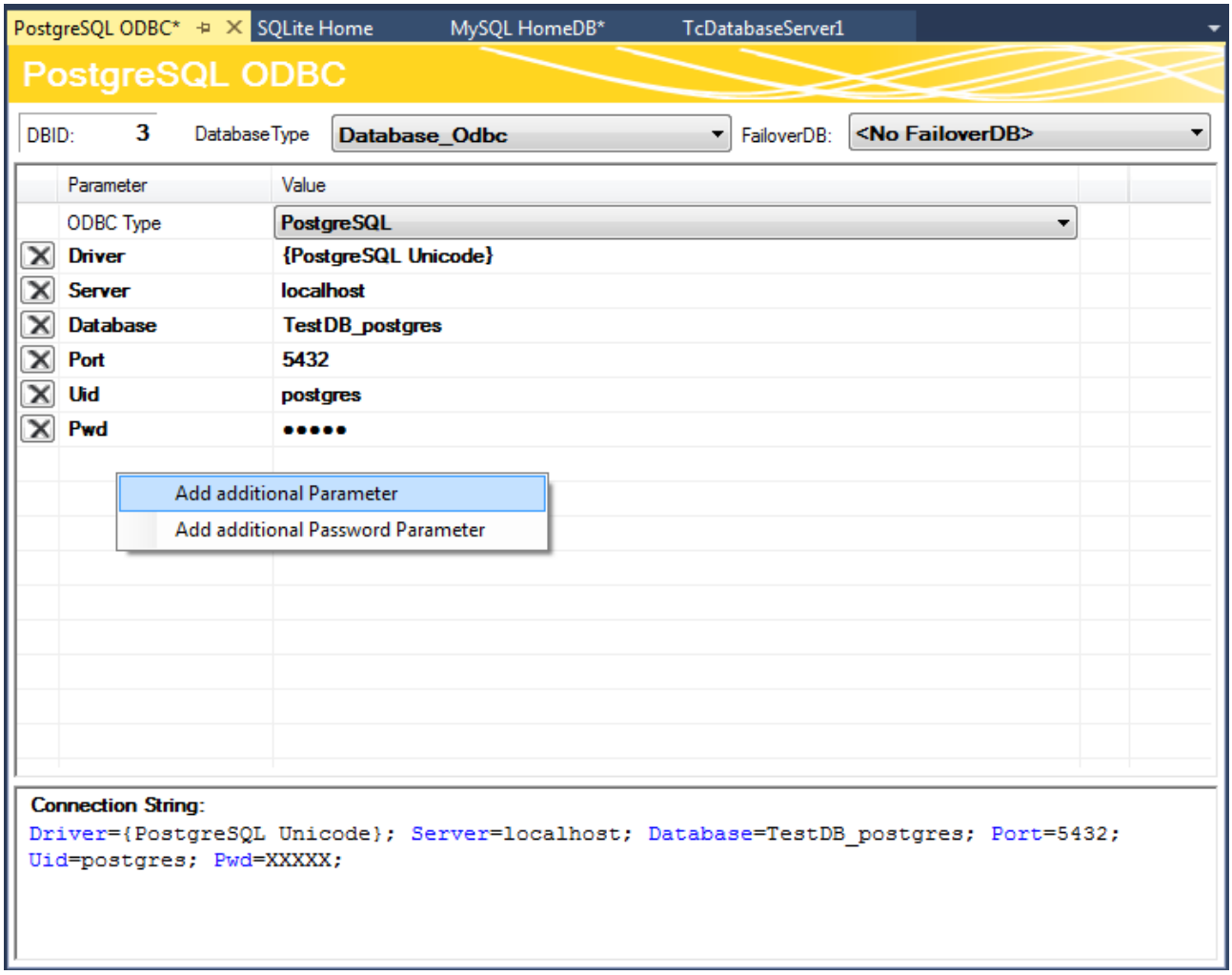


Im oberen Teil des Editors wird die Datenbank ID angezeigt (1), die für einige Funktionsbausteine in der SPS benötigt wird. Den Datenbanktyp der Zieldatenbank können Sie aus dem Drop-down-Menü auswählen (2). Hier können Sie auch die Odbc-Schnittstelle für eine Datenbank nutzen, die noch nicht unterstützt wird. Beachten Sie, dass je nach Datenbank nicht alle Funktionen des TwinCAT Database Servers gewährleistet werden.

Außerdem können Sie optional eine sogenannte FailOver-Datenbank (3) auswählen, welche im ‚Configure'-Mode beim Fehlerfall einspringt. Bei einem Verbindungsabbruch zum Netzwerk kann in einem solchen Fall automatisch gewährleistet werden, dass keine Daten verloren gehen und an anderer Stelle gespeichert werden.

Für jede Datenbank [▶ 126] stehen zusätzlich weitere einstellbare Parameter zur Verfügung. Je nach Datenbank wird ein Connection String (5) erstellt, welcher die Verbindung zur Datenbank beschreibt. Diese Anzeige dient zur Transparenz Ihrer eingestellten Parameter.

Über die Schaltfläche **CREATE** (4) können Sie eine neue Datenbank erstellen. Diese Funktion wird nur angezeigt, wenn sie von der jeweiligen Datenbank unterstützt wird.

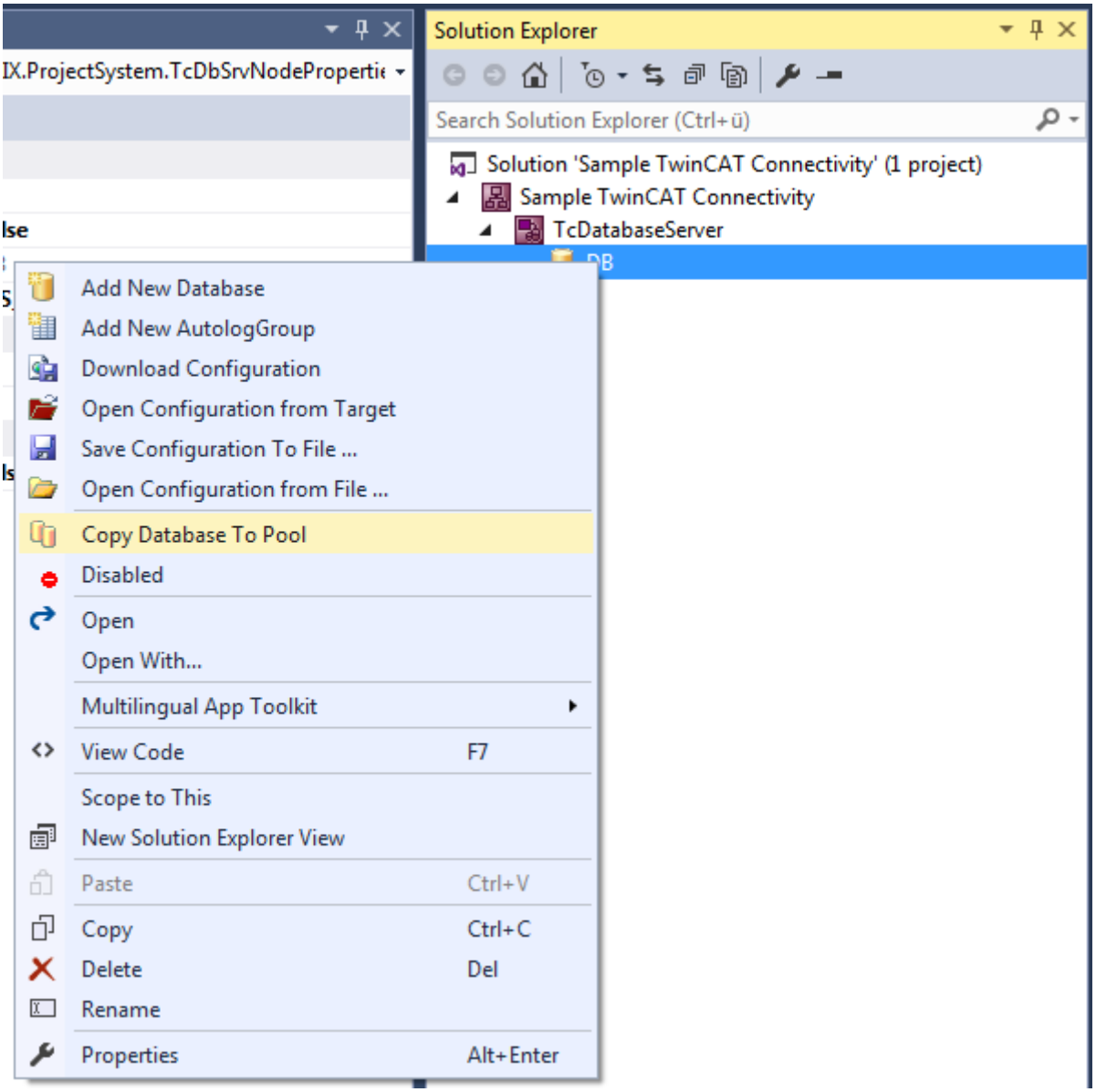


Sie können auch unbekannte Datenbanken mit einer ODBC-Schnittstelle konfigurieren. Dafür wählen Sie im in der Drop-down-Liste **ODBC Type** den Eintrag „Unknown Database" und fügen über die Befehle im Kontextmenü Parameter hinzu. Diese können auch Passwörter beinhalten, welche dann verschlüsselt abgespeichert werden. Daraus kann der gewünschte Connection String zusammengestellt werden. Beachten Sie, dass nur begrenzte Funktionen des TwinCAT Database Servers genutzt werden können. Nur die expliziten Funktionsbausteine des SQL Expert Modes werden unterstützt.

Die zusätzlichen Parameter können Sie nur über den Editor und nicht über das Eigenschaftenfenster einstellen.

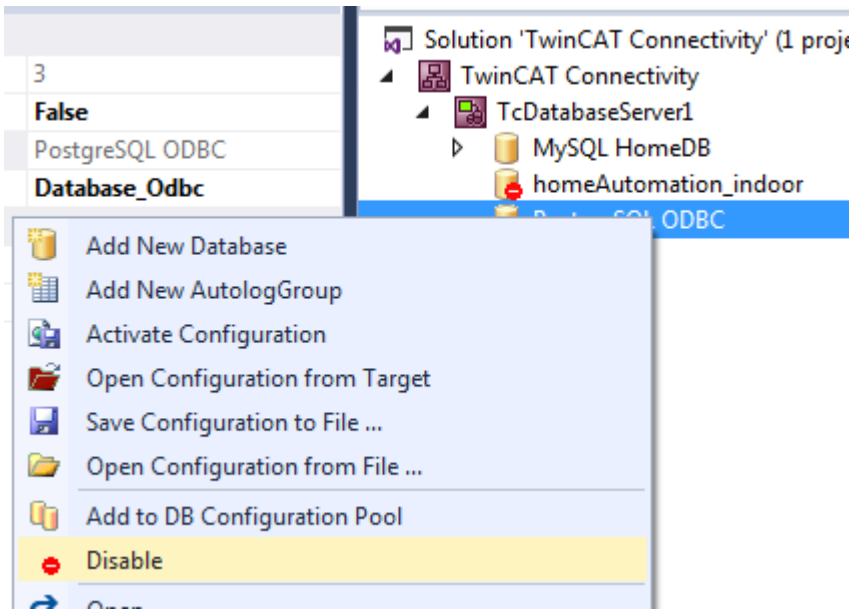
Kopieren einer Datenbankkonfiguration in den Datenbankpool

Für das Kopieren einer Datenbankkonfiguration in den Datenbankpool [▶ 51] steht ein entsprechendes Kommando im Kontextmenü zur Verfügung. Auch per drag-and-drop können Sie Datenbankkonfigurationen zwischen dem Projekt und dem Datenbankpool austauschen.



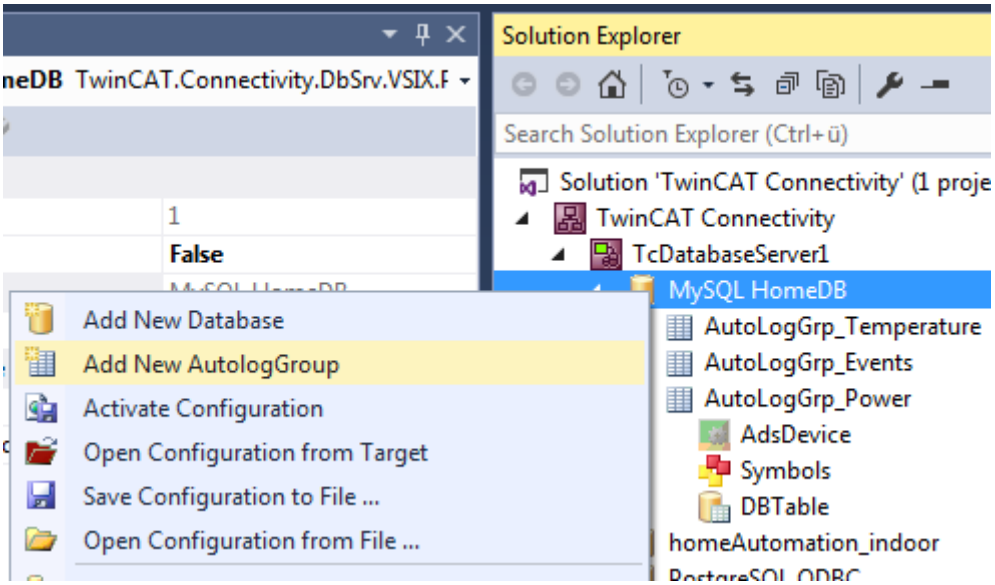
Datenbankkonfigurationen deaktivieren

Sie können einzelne Datenbankkonfigurationen im Projekt deaktivieren. Diese werden dann mit einer roten Markierung versehen und werden beim Aktivieren des Projektes ignoriert.

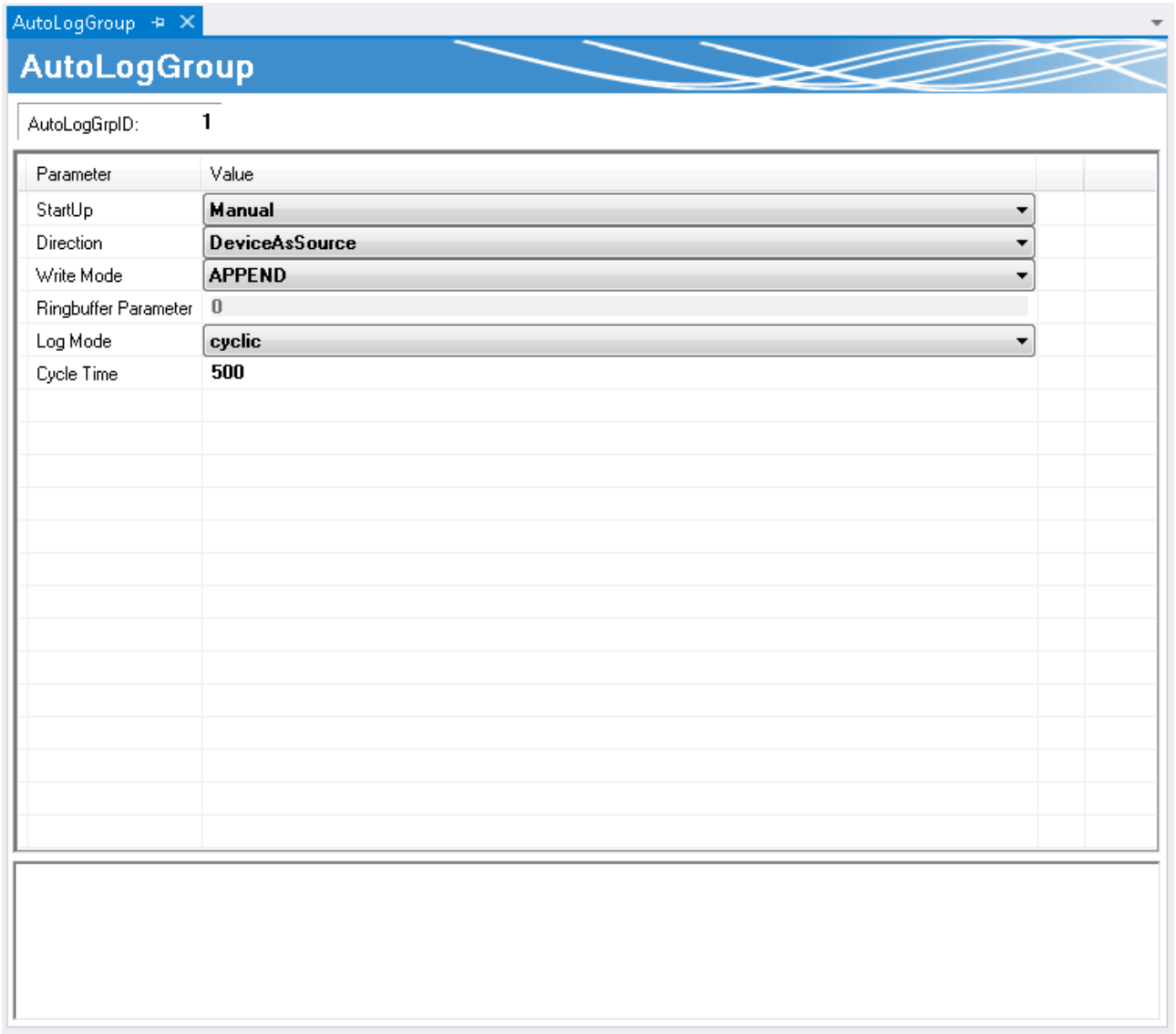


5.1.1.1.4 AutoLog-Gruppen konfigurieren

Eine neue AutoLog-Gruppe für die Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new AutologGroup** im Kontextmenü einer Datenbankkonfiguration oder über die Toolbar hinzufügen. Diese AutoLog-Gruppen beziehen sich auf die übergeordnete Datenbank.



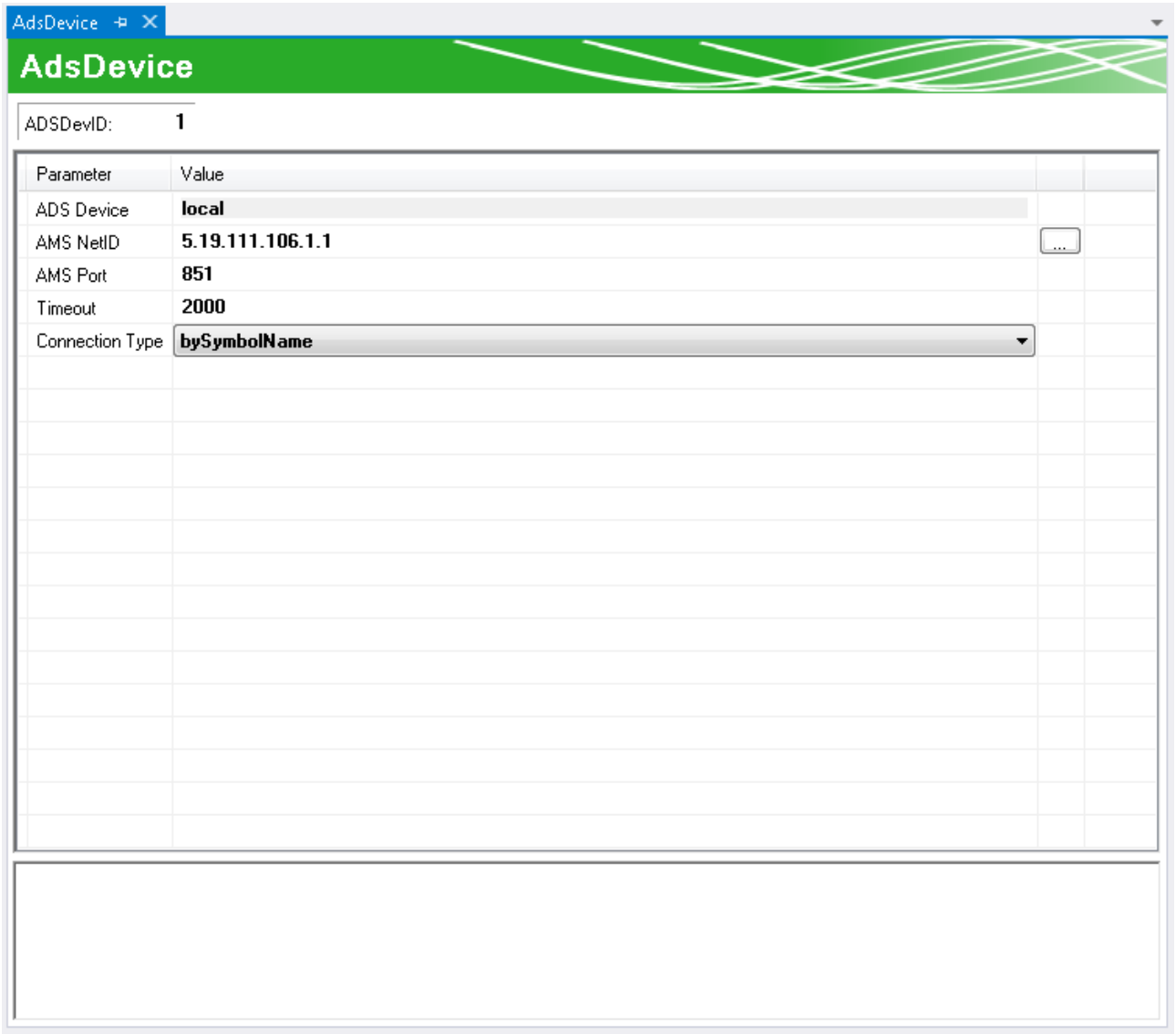
Eine neue AutoLog-Gruppe sowie die dazugehörigen Komponenten werden als Dateien im Projektordner hinzugefügt und im Projekt eingebunden. Dazu gehören das Ads Device, die Symbolgruppen und die Tabelleneinstellungen. Um diese Dateien im Projekt zu speichern, sollten Sie die TwinCAT-Connectivity-Projektdatei speichern. Diese Dateien können Sie dann in Editoren oder im Eigenschaftenfenster bearbeiten.



StartUp	Der AutoLog-Modus kann über den manuellen Weg (durch einen Befehl in der SPS oder aus dem Konfigurator) oder automatisch beim Systemstart eingeschaltet werden.
Direction	Das eingestellte ADS-Gerät dient als Datenziel oder als Datenquelle.
Write Mode	Die Daten können in einer Datenbank zeilenweise angehängt, auf zeitlicher Basis oder nach Anzahl in einem Ringpuffer gehalten oder einfach an der entsprechenden Position aktualisiert werden.
Ringbuffer Parameter	Je nach Einstellung stellt dieser Parameter die Zeit oder die Zyklen dar, nach der der Ringbuffer aktualisiert wird.
Log Mode	Entweder wird die Variable nach Ablauf eines gewissen Zyklus oder auf Änderung geschrieben.
Cycle Time	Zykluszeit, nach welcher die Variable geschrieben wird.

Ads Device konfigurieren

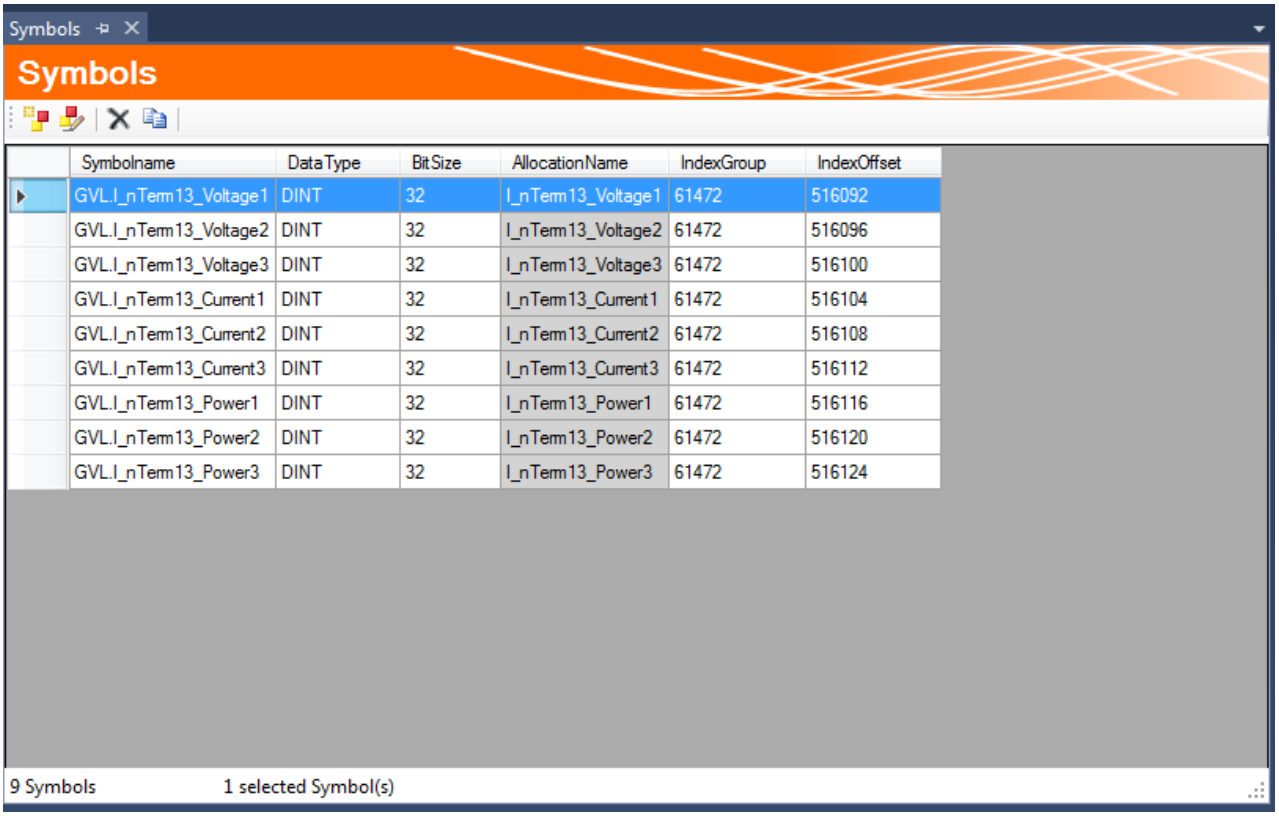
Das Ads Device wird automatisch unter eine AutoLog-Gruppe angelegt. Das Ads Device ist im häufigsten genutzten Anwendungsfall die SPS-Laufzeit. Folgende Einstellungen können im Editor getroffen werden:



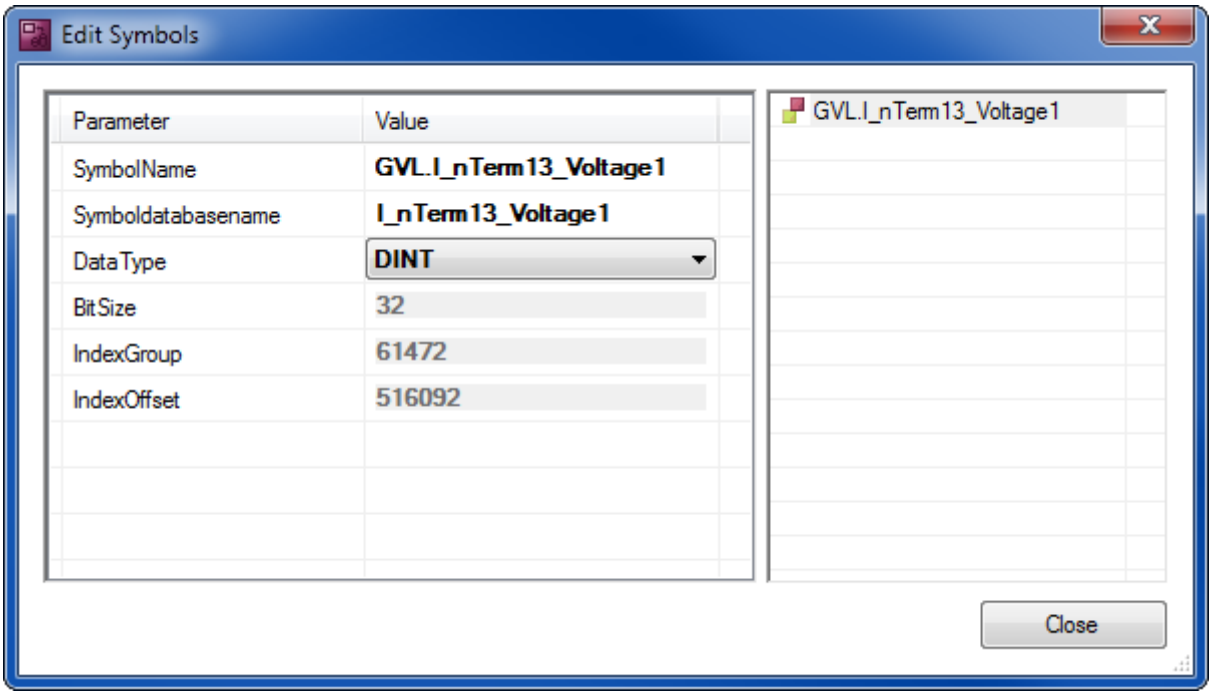
ADS Device	Bezeichnung des ADS-Zielgeräts.
AMS NetID	Adresse des Zielgeräts im TwinCAT-Netzwerk.
AMS Port	Port des Zielgeräts im TwinCAT-Netzwerk.
Timeout	Zeit, nach der von einem Verbindungsabbruch zum Zielgerät ausgegangen wird.
Connection Type	bySymbolName: Verbindung wird anhand des Namens des Symbols hergestellt. byIndexGroup: Verbindung wird anhand des Speicherindex hergestellt.

Symbole konfigurieren

Je nachdem, ob das ADS-Gerät Datenziel oder Datenquelle ist, werden die Symbole, die Sie hier einstellen, in die Datenbank geschrieben oder aus der Datenbank ausgelesen. Für einen komfortablen Zugriff können Sie den TwinCAT Target Browser [▶ 52] verwenden. Hier können die Symbole auf dem Target gesucht und per drag-and-drop zwischen den beiden Tools kommuniziert werden.



Sie könne Symbole auch manuell zur Symbolgruppe hinzufügen oder bearbeiten. Je nachdem, ob im ADS-Gerät der Verbindungstyp über die Symbolnamen oder die Index-Gruppen ausgewählt wurden, werden entsprechende Informationen benötigt. Dabei wird immer vom ADS-Gerät ausgegangen.



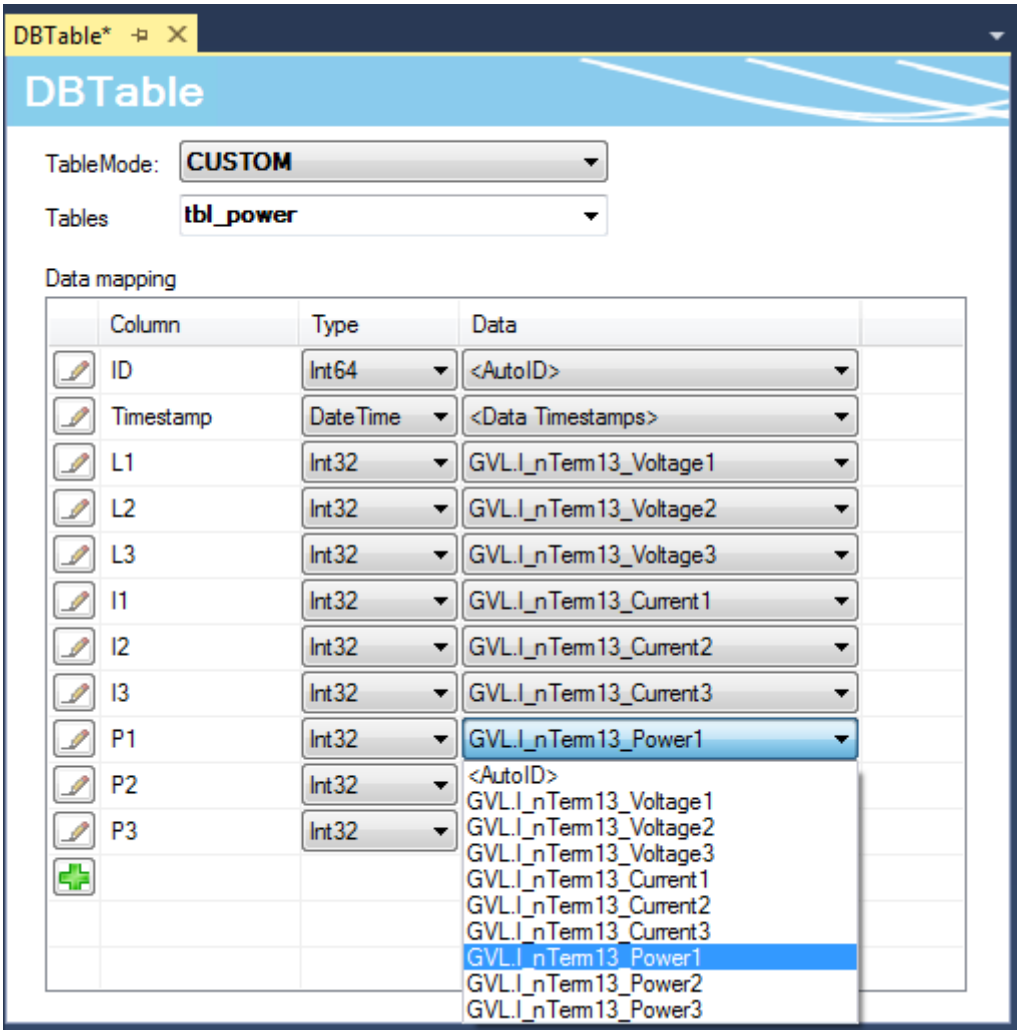
SymbolName	ausgehend vom einstellten ADS-Gerät wird das Symbol angesprochen
Symboldatabasename	Name der Variable in der Datenbanktabelle
DataType	SPS-Datentyp des Symbols
BitSize	Bit-Größe des Symbols (wird bei automatisch für die Datentypen eingestellt)
IndexGroup	Indexgruppe im TwinCAT-System
IndexOffset	Indexoffset im TwinCAT-System

Tabelle konfigurieren

Die Tabelle in einer Datenbank kann nach einer Standardtabellenstruktur oder nach einer individuellen Struktur aufgebaut sein.

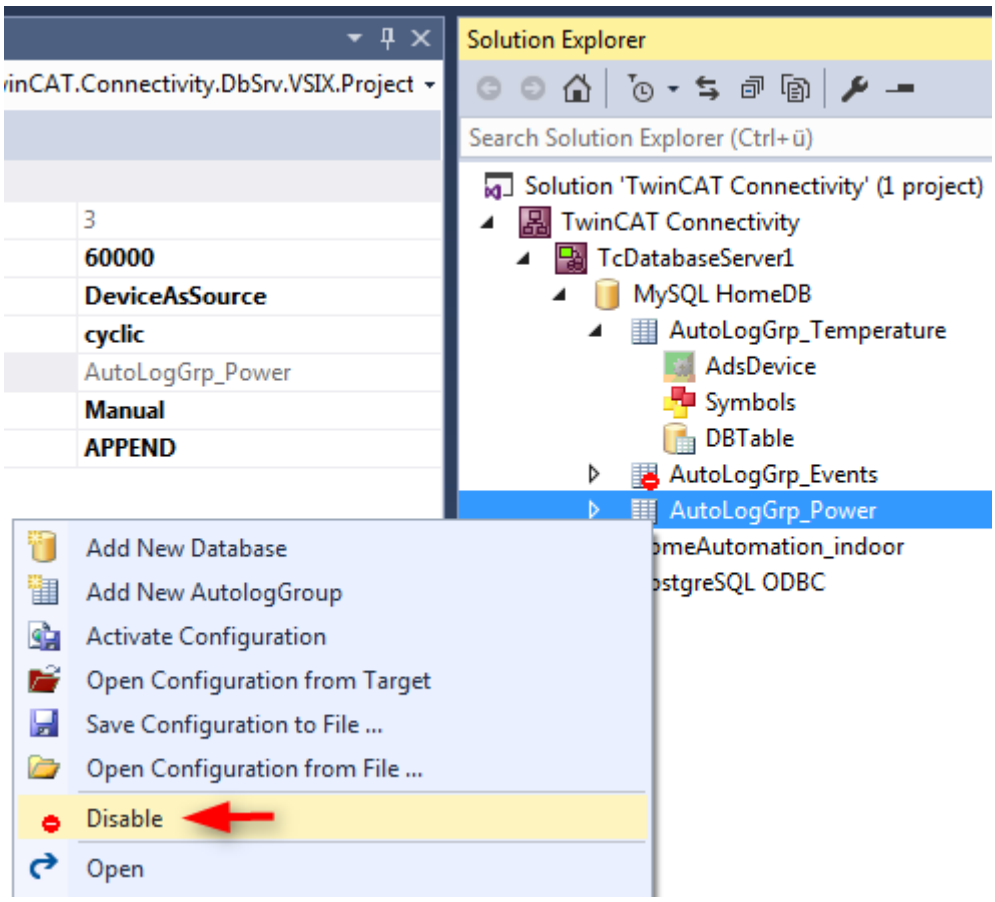
Die entsprechende Tabelle können Sie aus einer Liste möglicher Tabellen auswählen. Ist die Tabelle noch nicht vorhanden, können Sie diese mithilfe des SQL Query Editors erzeugen. Falls Sie die Standardtabellenstruktur auswählen, zeigt Ihnen ein blauer Haken an, ob die ausgewählte Tabelle dieser Struktur entspricht.

Der spezifische Tabellentyp bietet Ihnen die Möglichkeit, die einzelnen Symbole, welche in der Symbolgruppe eingestellt wurden, auf die Tabellenspalten in der Datenbank beliebig zu verteilen. Wird ein Datensatz nun während des AutoLog-Modus in die Datenbank geschrieben, werden die aktuellen Werte der Symbolgruppe zum Abtastzeitpunkt in der entsprechenden Spalte der Tabelle gespeichert.



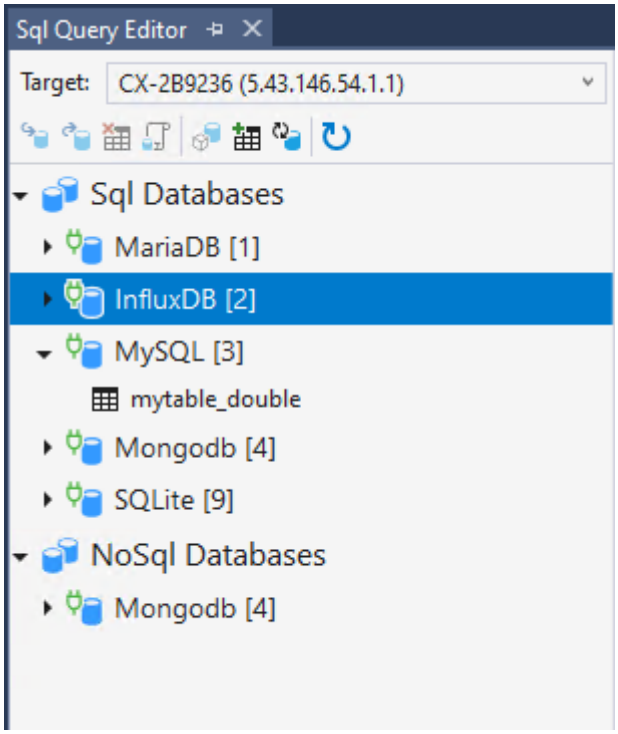
Deaktivieren von AutoLog-Gruppen

Genauso wie einzelne Datenbankkonfigurationen können Sie auch einzelne AutoLog-Gruppen im Projekt deaktivieren. Diese werden beim Aktivieren des Projektes auf dem Zielsystem ignoriert. Eine deaktivierte AutoLog-Gruppe wird mit einer roten Markierung versehen und kann mit demselben Befehl wieder aktiviert werden.



5.1.1.1.5 SQL Query Editor

Der SQL Query Editor ist ein Tool des Database Servers, um die Entwicklung Ihrer Applikation zu unterstützen. Mit dem Tool können Verbindungen und SQL-Befehle getestet und die Kompatibilität zwischen SPS und Datenbanken geprüft werden.



Nachdem der TwinCAT Database Server des Zielsystems gewählt wird, lädt der SQL Query Editor die aktuelle Datenbankkonfiguration und die Tabellen der erfolgreich verbundenen Datenbanken. Je nachdem, ob die Datenbank die SQL und die NoSQL-Schnittstelle (vom TwinCAT Database Server) unterstützt, wird sie unter der jeweiligen Kategorie aufgeführt.

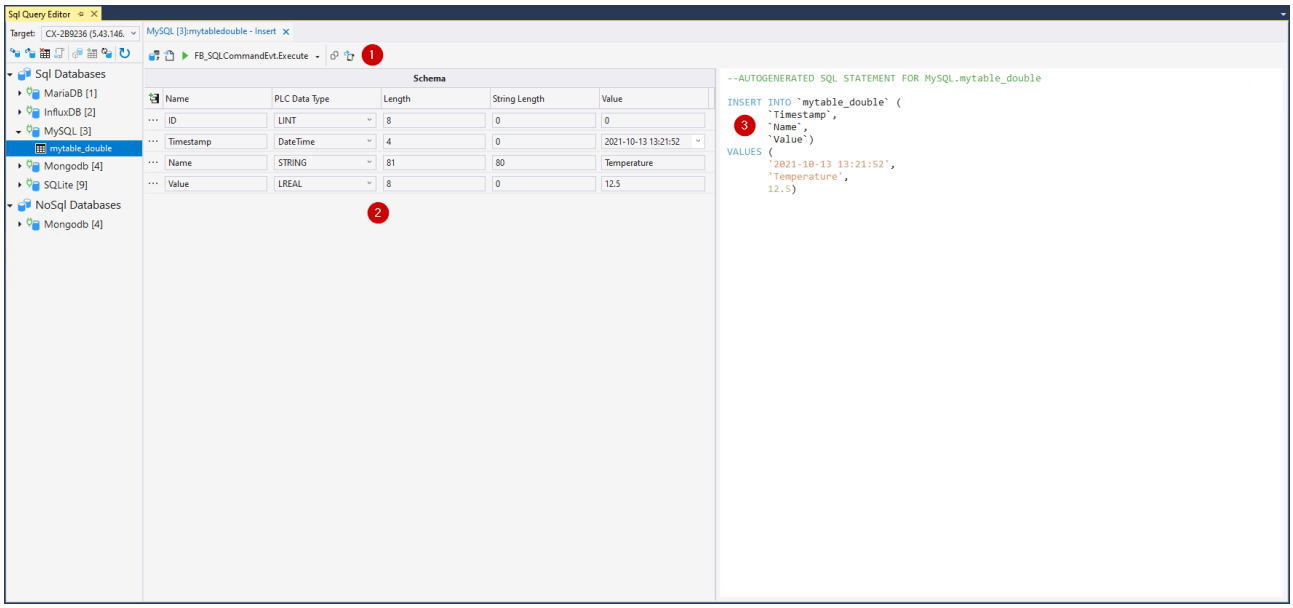
Unter der Auswahl des Zielsystems befindet sich eine Statusleiste mit den verfügbaren Befehlen:

Tabellenebene	
Insert Arbeitsbereich	Öffnet den Insert-Arbeitsbereich, um Datensätze mit SQL in die ausgewählte Tabelle zu schreiben.
Select Arbeitsbereich	Öffnet den Select-Arbeitsbereich, um Datensätze mit SQL aus der gewählten Tabelle zu lesen.
Delete/Drop Arbeitsbereich	Öffnet den Delete/Drop-Arbeitsbereich, um Datensätze mit SQL aus der gewählten Tabelle zu löschen oder ganze Tabellen zu löschen.
NoSQL Arbeitsbereich	Öffnet den NoSQL-Arbeitsbereich, um NoSQL-spezifische Abfragen auszuführen.
Datenbankebene	
Stored Procedure Arbeitsbereich	Öffnet den Stored Procedure-Arbeitsbereich, um gespeicherte Prozeduren der Datenbank auszuführen.
Tabellen Arbeitsbereich	Öffnet den Tabellen Arbeitsbereich, um neue Tabellen in der ausgewählten Datenbank zu erstellen.
Tabellen aktualisieren	Aktualisiert die verfügbaren Tabellen der ausgewählten Datenbank.
Allgemein	
Datenbanken aktualisieren	Aktualisiert den gesamten Datenbankbaum.

Die Arbeitsbereiche werden rechts neben dem Baum unter dem entsprechenden Reiter geöffnet. Auch von der gleichen Tabelle können mehrere Reiter zu einem Zeitpunkt geöffnet werden.

Insert-Arbeitsbereich

Der Insert-Arbeitsbereich ermöglicht das Schreiben von Daten in die ausgewählte Tabelle über die TwinCAT Database Server Schnittstelle für SQL-Funktionsbausteine.



Im unteren Bereich (2) befindet sich eine Tabelle mit den einzelnen Datensymbolen im zu schreibenden Datensatz. Der Name, PLC-Datentyp, die Bytelänge sowie der Wert können hier bestimmt werden. Die eingetragenen Werte werden anschließend über den Befehl zum Generieren der SQL-Anweisung verwenden.

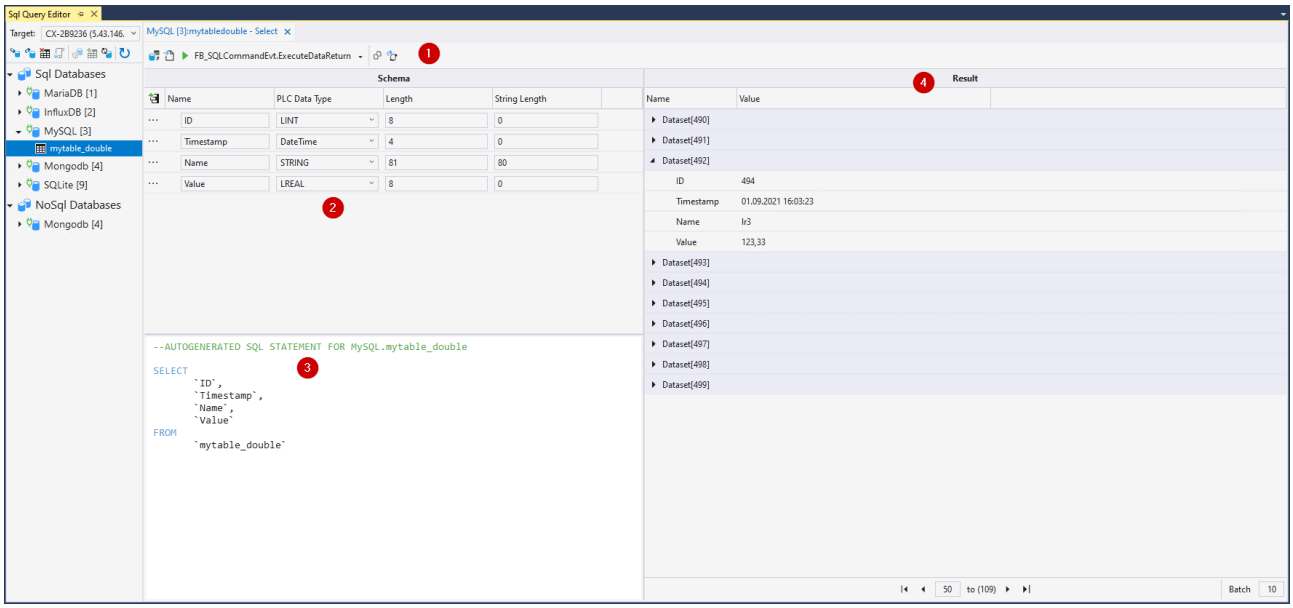
Diese SQL-Anweisung wird dann in einem Textfeld (3) zur Verfügung gestellt. Je nach Syntax der Datenbank kann der Inhalt unterschiedlich ausfallen.

In der oberen Statusleiste befinden sich die Kommandos zum Interagieren mit dem TwinCAT Database Server (1).

Kommando	Beschreibung
Lese Tabellen Schema	Liest das Tabellenschema der Tabelle des Arbeitsbereichs aus.
Generiere SQL-Anweisung	Generiert abhängig von der Datenbanksyntax aus der vorliegenden Tabelle die SQL-Anweisung.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die im Textfeld (3) stehende Anweisung aus.
Kopieren der Anweisung	Kopiert die im Textfeld (3) stehende Anweisung als TwinCAT kompatible Syntax.
Exportieren als Struktur	Exportiert die Struktur der Tabelle der Eingabewerte zu einem TwinCAT 3 kompatiblen DUT.

Select-Arbeitsbereich

Der Select-Arbeitsbereich ermöglicht das Lesen von Daten in die ausgewählte Tabelle über die TwinCAT Database Server Schnittstelle für SQL-Funktionsbausteine.



Im unteren Bereich (2) befindet sich eine Tabelle mit den einzelnen Datensymbolen im zu lesenden Datensatz. Der Name, PLC-Datentyp, sowie die Bytelänge können hier bestimmt werden. Diese Informationen werden anschließend zum Interpretieren der Daten benötigt.

Diese SQL-Anweisung wird dann in einem Textfeld (3) zur Verfügung gestellt. Je nach Syntax der Datenbank kann der Inhalt unterschiedlich ausfallen.

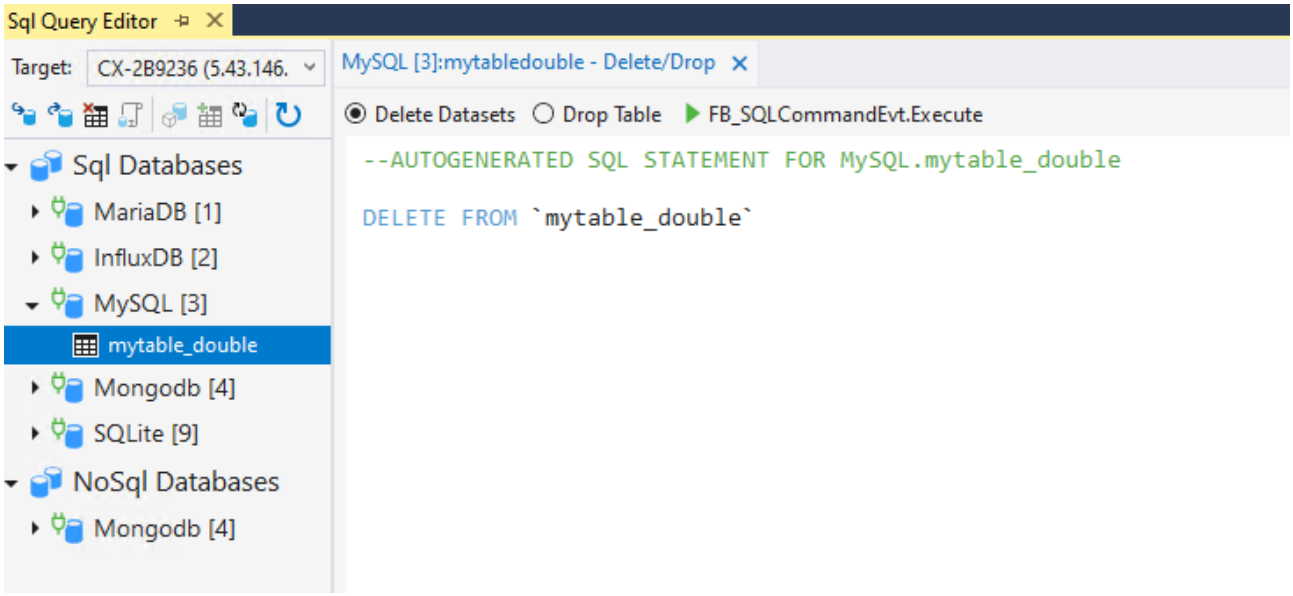
Im Ergebnis-Feld (4) werden nach Ausführung der Anweisung die Daten angezeigt. Falls mehrere Ergebnisse zurückgeliefert werden, können diese unter über die Seiten durchgeschaltet werden.

In der oberen Statusleiste befinden sich die Kommandos zum Interagieren mit dem TwinCAT Database Server (1).

Kommando	Beschreibung
Lese Tabellen Schema	Liest das Tabellenschema der Tabelle des Arbeitsbereichs aus.
Generiere SQL-Anweisung	Generiert abhängig von der Datenbanksyntax aus der vorliegenden Tabelle die SQL-Anweisung.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die im Textfeld (3) stehende Anweisung aus.
Kopieren der Anweisung	Kopiert die im Textfeld (3) stehende Anweisung als TwinCAT kompatible Syntax.
Exportieren als Struktur	Exportiert die Struktur der Tabelle der Eingabewerte zu einem TwinCAT 3 kompatiblen DUT.

Delete/Drop-Arbeitsbereich

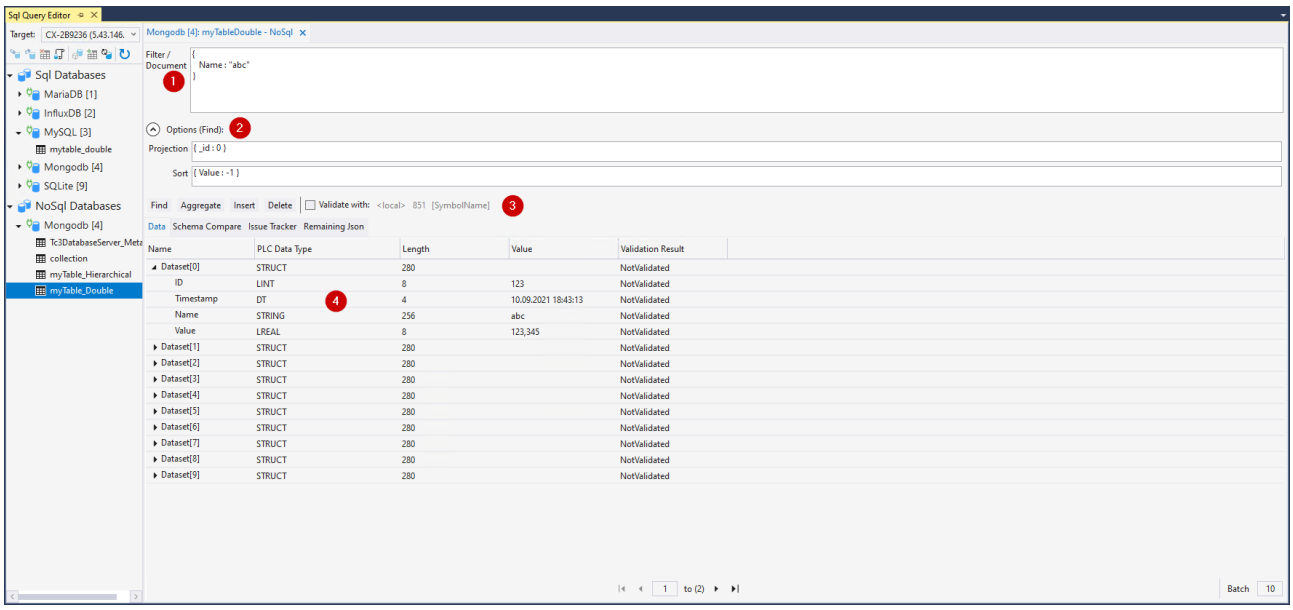
Der Delete/Drop-Arbeitsbereich bietet die Option SQL-Anweisungen abzusetzen, um entweder Daten aus einer Tabelle oder die gesamte Tabelle aus der Datenbank zu löschen.



Hierfür kann zwischen den beiden Optionen in der Statusleiste ausgewählt werden. Die der Datenbank entsprechenden Syntax wird daraufhin im Anweisungsfeld generiert. Um diese mit der TwinCAT Database Server Schnittstelle auszuführen, steht der Schalter **FB_SQLCommandEvt.Execute** zur Verfügung.

NoSql-Arbeitsbereich

Der NoSql-Arbeitsbereich unterstützt die speziellen Funktionen von NoSql-Datenbanken bzw. der TwinCAT Database Server NoSQL-Schnittstelle.



ID	Bezeichnung	Funktion
1	Filter/Document	Je nachdem welche Funktion genutzt wird, agiert dieses Eingabefeld als Dokument oder als Filter im Json-Format. Falls Sie einen Find ausführen möchten und zusätzlich eine Projektion oder eine Sortierung vornehmen möchten, lassen sich mit den darunterliegenden Options(Find) diese Felder füllen.
2	Options (Find)	Beschreibt zusätzliche Parameter für die Find-Funktion, wie die Projektion oder Sortierung.
3	Steuerelemente	Steuerelemente zur Interaktion mit der TwinCAT Database Server Schnittstelle für NoSQL.
4	Datenanzeige	Auflistung von zurückgelieferten Daten. Die Navigation ermöglicht die Iteration durch die verfügbaren Seiten.

Find: Führt eine Suchabfrage mit dem im Textfeld (1) eingetragenen Filter aus. Optional kann ebenfalls über die Options(Find) -Felder eine Projektion oder eine Sortierung vorgenommen werden. Hierbei werden Daten zurückgeliefert und in der Datenanzeige (4) aufgeführt. Die Syntax der Filter ist dabei datenbankspezifisch.

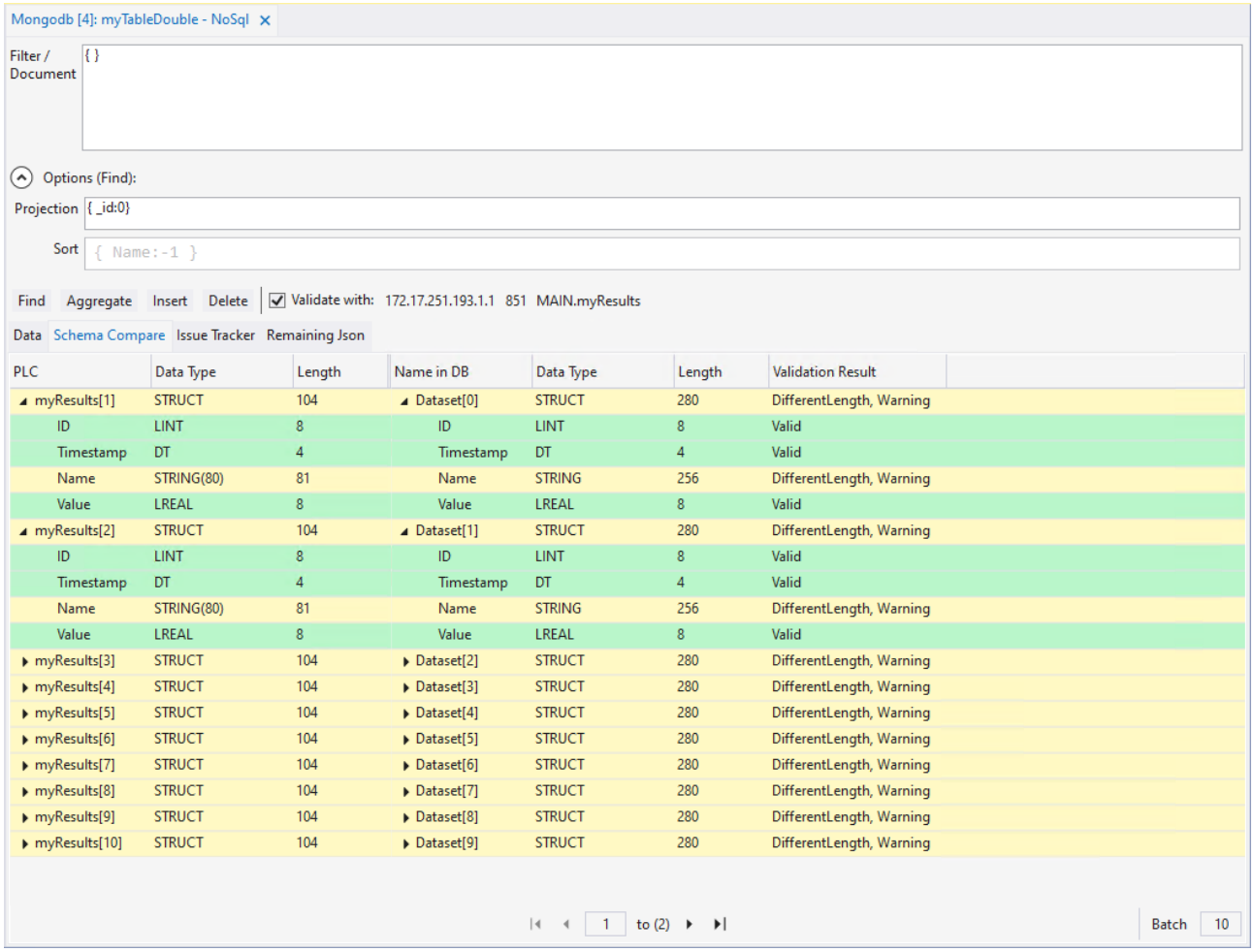
Aggregate: Führt eine Aggregation mit dem im Textfeld (1) eingetragenen Parametern aus. Hierbei werden Daten zurückgeliefert und in der Datenanzeige (4) aufgeführt. Die Syntax der Filter ist dabei datenbankspezifisch.

Insert: Führt eine Insert-Abfrage des im Textfeld (1) eingetragenen (Json-)Dokuments oder Dokument-Array aus. Diese werden dann in die Collection geschrieben.

Delete: Führt eine Delete-Abfrage auf die Daten aus, welche mit dem Filter im Textfeld (1) gefunden wurden. Die gefundenen Daten werden aus der Collection gelöscht.

Validate: Wird diese Option angewählt werden die Datenabfragen nicht automatisch nach ihrem eigenen Schema geparst, sondern versucht diese Daten auf die Struktur des Symbols aus der SPS abzubilden, welches über diese Parameter angegeben wurde.

Bei letzterer Funktion kann eine Find-Abfrage zu Konflikten führen. Im Gegensatz zu Strukturen im SPS-Prozessabbild, müssen Datensätze in NoSql-Datenbanken keinem festen Schema folgen. Möglicherweise besitzen abgefragte Dokumente keine Daten zu einem bestimmten Element in der SPS-Struktur. Oder der Datensatz trägt Daten, welche nicht in der SPS-Struktur vorhanden sind. Zugeordnet werden diese Daten über den Namen bzw. dem Attribut „ElementName“ in der SPS.



In der Registerkarte **Schema Compare** können die Unterschiede der Daten nachvollzogen werden. Im obigen Beispiel ist zu erkennen, dass im Falle des zurückgelieferten Dokuments in der PLC-Struktur die Variable „Name“ eine andere Datentyplänge hat als die der Datenbank. Die entsprechenden Farben zeigen die Gewichtung des Konflikts:

Rot: Es sind zu viele oder zu wenige Daten vorhanden.

Gelb: Die Bytelänge des Datensatzes stimmt nicht überein oder darunterliegende Datensätze sind übrig oder nicht vorhanden.

Grün: keine Konflikte

Diese Konflikte werden auch als Liste unter der Registerkarte **Issue Tracker** aufgeführt. Sie kann auch als String-Array bei Bedarf in die SPS eingelesen werden.

In der Registerkarte **Remaining Json** werden übrig gebliebene Datensätze als Json zurückgegeben. Auch diese Informationen können als String in die SPS gelesen werden.

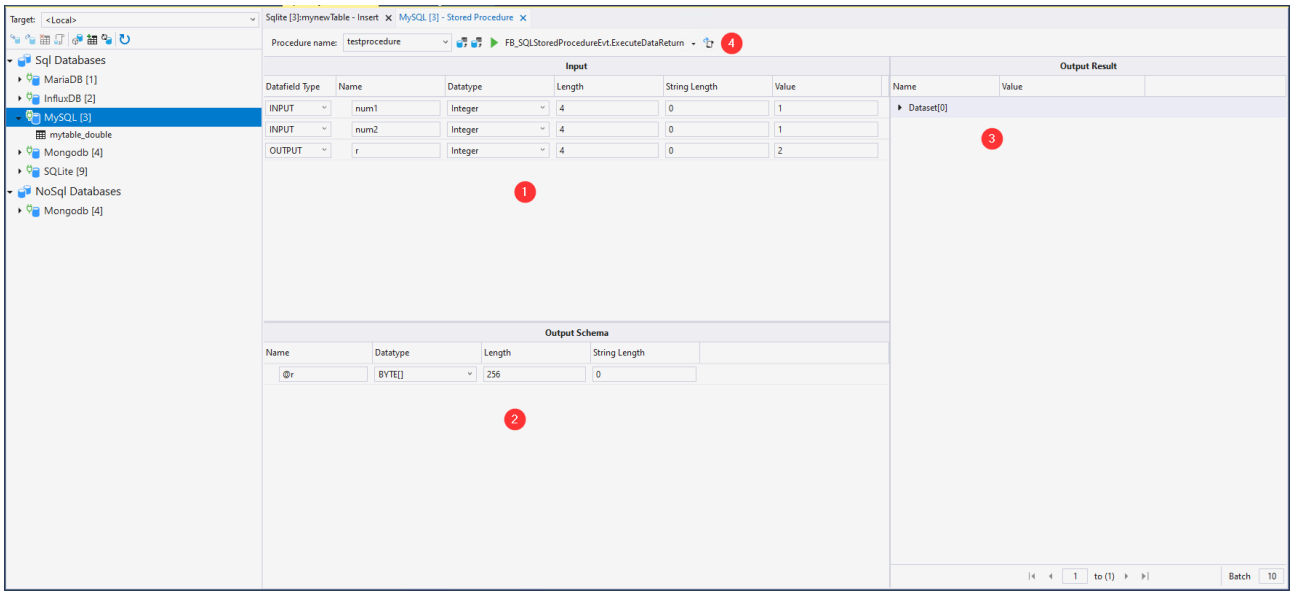
Über die Steuerelemente in der Statusleiste kann, wie von den anderen Datendarstellungen bekannt, durch die Daten iteriert werden. Dabei kann die Menge der gleichzeitig angezeigten Datensätze angegeben werden.

Stored Procedure-Arbeitsbereich

Der TwinCAT Database Server unterstützt „Stored Procedures“, die viele Datenbanken bereitstellen, um komplexere Abfragen auf der Datenbankebene zu verarbeiten oder eine vereinfachte Schnittstelle zur Verfügung zu stellen.

Falls **Stored Procedures** in der Datenbank vorhanden sind, werden diese in der Dropdown-Liste der Statusleiste (4) aufgeführt.

Darunter befindet sich die Tabelle für die Eingangsparameter (1), sowie für das Ausgabe-Schema (2). Zusätzlich gibt es eine Ansicht für die Ausgabeergebnisse (3). Bei erfolgreicher Ausführung der **Stored Procedure** werden die Ergebnisse hier angezeigt.

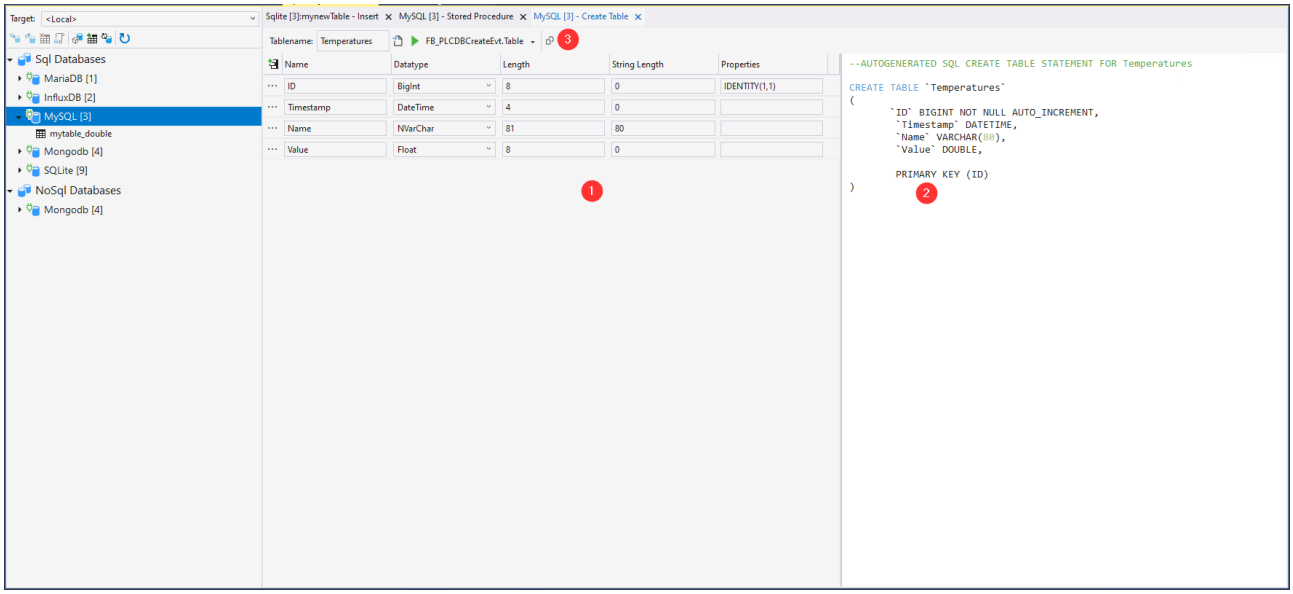


Die Statusleiste hat folgende Befehle:

Kommando	Beschreibung
Lese Stored Procedure Eingangsschema	Liest das Eingangsparameterschema aus. Die Ergebnisse werden in der Tabelle 1 aufgeführt.
Lese Stored Procedure Ausgangsschema	Liest das Ausgangsparameterschema aus. Die Ergebnisse werden in der Tabelle 2 aufgeführt. **Info:** Hierfür ist die Ausführung der Stored Procedure notwendig. Hierbei können je nach Programmierung Daten verändert werden.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die Stored Procedure aus.
Exportieren als Struktur	Exportiert die Struktur der Tabelle zu einem TwinCAT 3 kompatiblen DUT.

Table-Arbeitsbereich

Der Table-Arbeitsbereich dient zur Erstellung von neuen Tabellen.

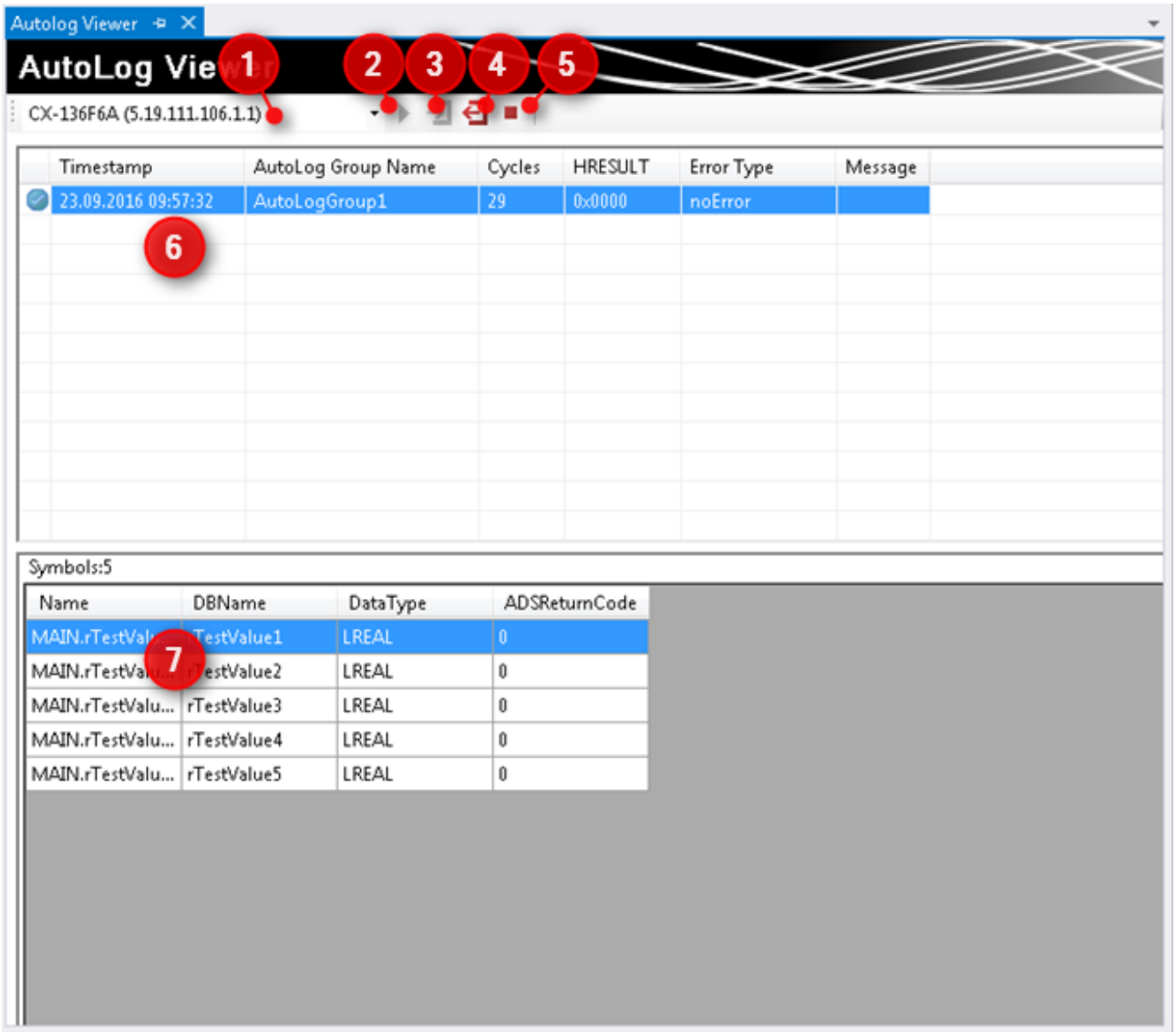


Hier kann die Tabellenstruktur (1) erstellt werden und daraus eine SQL-Anweisung im entsprechenden Feld (2) generiert werden. Hierfür kann die Statusleiste (3) mit folgenden Kommandos verwendet werden:

Kommando	Beschreibung
Tabellenname	Bestimmt den Tabellennamen der neuen Tabelle.
Generiere SQL-Anweisung	Generiert abhängig von der Datenbanksyntax aus der vorliegenden Tabelle die SQL-Anweisung.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die Stored Procedure aus.
Kopieren der Anweisung	Kopiert die im Textfeld (2) stehende Anweisung als TwinCAT kompatible Syntax.

5.1.1.1.6 AutoLog View

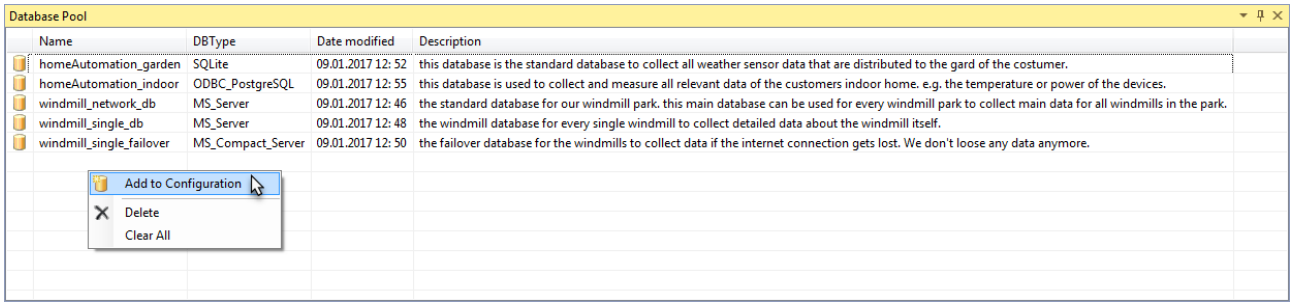
Der AutoLog Viewer des TwinCAT Database Server ist ein Tool, um den AutoLog-Modus zu steuern und zu überwachen. Ähnlich wie bei der TwinCAT SPS können Sie sich auf ein Zielsystem einloggen. Im eingeloggten Zustand kann der AutoLog-Modus gestartet oder gestoppt werden. Im unteren Bereich des Fensters werden Informationen über den aktuellen Zustands des Loggens mitgeteilt. Durch das Selektieren einer AutoLog-Gruppe werden weitere Informationen über die geloggten Symbole angezeigt.



ID	Bezeichnung	Funktion
1	Zielsystem	Auswahl des Zielsystems mit installiertem TwinCAT Database Server
2	Start	Manueller Start des AutoLog-Modus
3	Einloggen	Einloggen auf den aktiven AutoLog-Prozess
4	Ausloggen	Ausloggen aus dem aktiven AutoLog-Prozess
5	Stopp	Manueller Stopp des AutoLog-Modus
6	Autologgruppen	Auflistung konfigurierter Autolog-Gruppen auf dem Zielsystem
7	Symbole	Auflistung konfigurierter Symbole der ausgewählten AutoLog-Gruppe

5.1.1.1.7 Database Configuration Pool

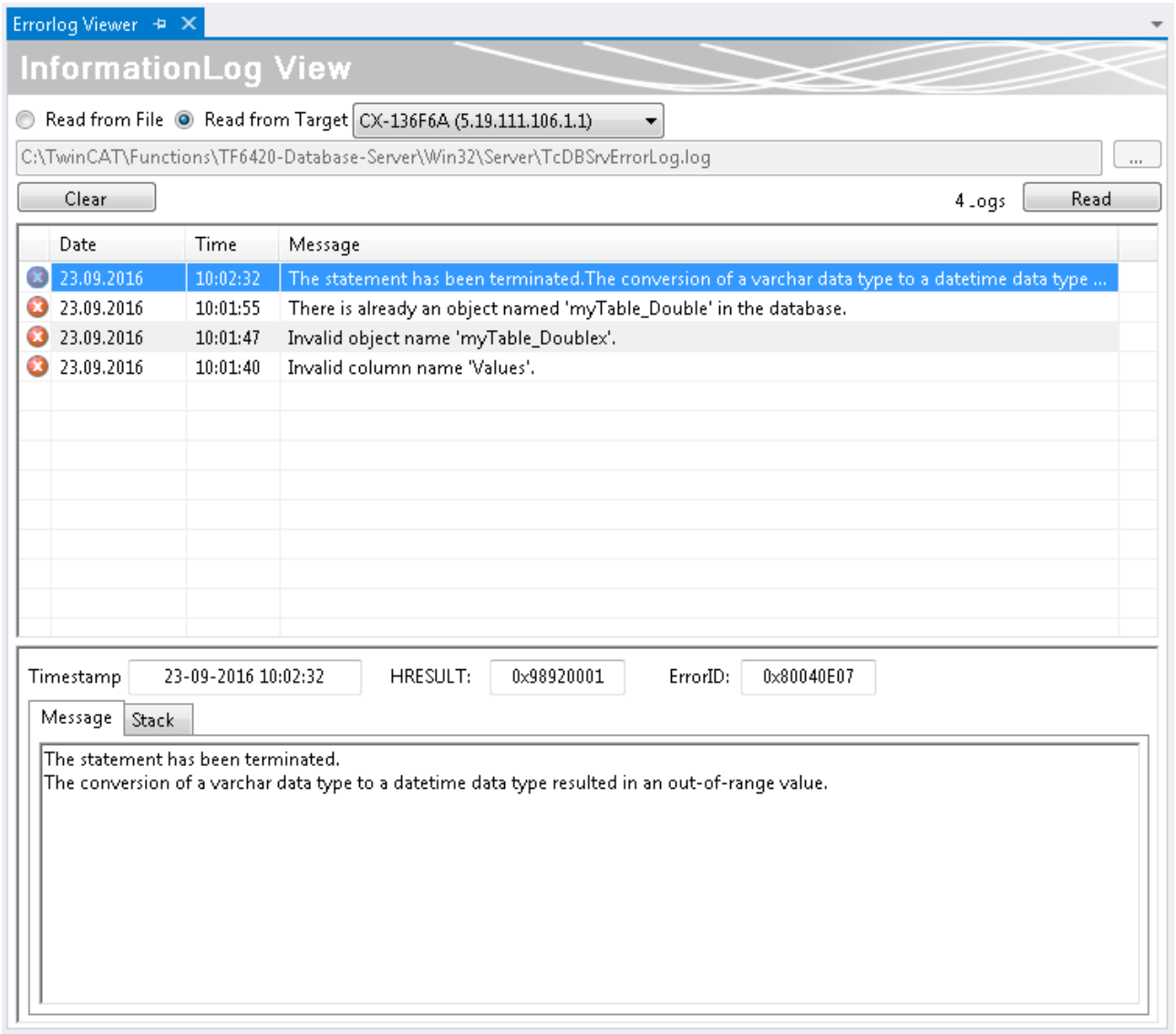
Der Database Configuration Pool ist ein globaler Ablageort für Datenbankkonfigurationen auf dem Entwicklungssystem. Er dient dem Entwickler als Speicherort für projektungebundene Datenbankkonfigurationen oder Vorlagen für mehrfach verwendete Konfigurationen. Dieser Pool hat sowohl für die Visual-Studio-Integration als auch für den Standalone-Konfigurator denselben benutzerspezifischen Speicherort. Bei der Deinstallation vom TwinCAT Database Server bleiben die Dateien bestehen.



5.1.1.1.8 InformationLog View

Der InformationLog View ist ein Tool, um die Logdateien vom TwinCAT Database Server auszulesen. Protokollierte Informationen werden mit einem Zeitstempel, IDs und Fehlermeldungen im Klartext angezeigt.

Die Log-Dateien können nicht nur über den direkten Dateizugriff eingesehen oder geleert werden, sondern auch direkt über das Target. Gerade für verteilte Database Server im Netzwerk ist dies vorteilhaft, um einen schnellen und einfachen Zugriff auf die Logdatei zu erlangen. Für diesen Zugriff muss eine Route zum Zielgerät bestehen.

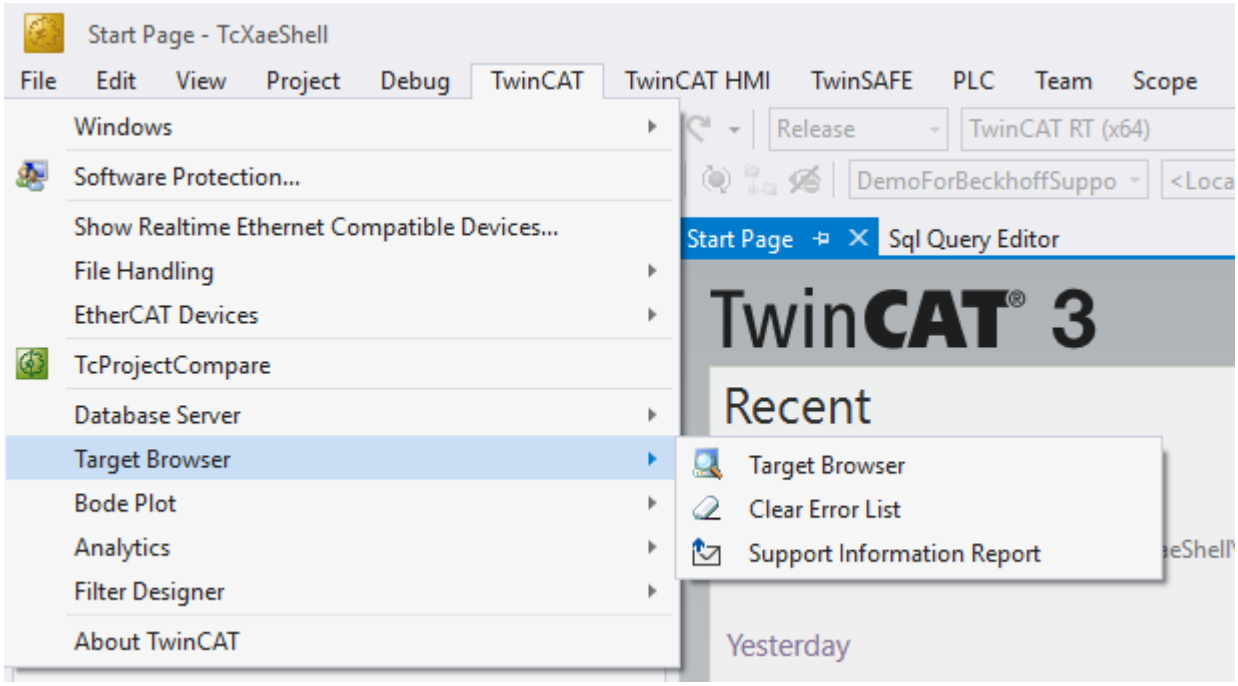


5.1.1.1.9 TwinCAT Target Browser

Der TwinCAT Target Browser ist die zentrale Datenverwaltungsschnittstelle im TwinCAT 3 Engineering. Er kann Daten von verschiedenen TwinCAT-Zielsystemen live über ADS bereitstellen oder auch auf historische Daten in Datenbanken zugreifen. Je nachdem welche TwinCAT 3 Functions auf einem Engineering-System installiert sind, wird die ADS-Standard-Extension des TwinCAT Target Browsers um weitere Extensions erweitert (siehe Extensions [▶ 53]).

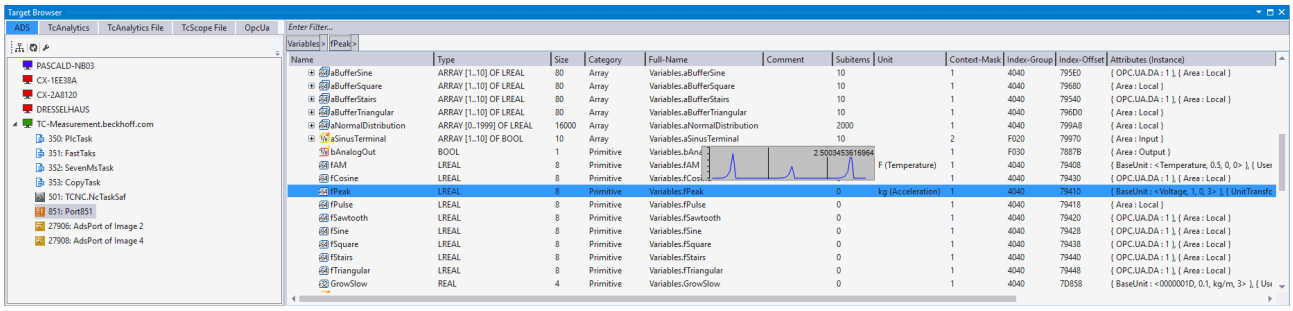
Aufruf Optionen

Der Target Browser über das Scope Menu im Visual Studio® aufrufbar. Aufgrund der immer größeren Verbreitung im TwinCAT System ist er auch über das TwinCAT Menü unter dem Eintrag **Target Browser** zu öffnen.



Darüber hinaus ist es möglich, dass die verschiedenen Tools, die den Target Browser verwenden, weitere Aufrufe zur Verfügung stellen. Beispielsweise machen dies einige Produkte über das Kontextmenü auf ihrem jeweiligen Projekt-Knoten.

Architektur



Der TwinCAT Target Browser ist innerhalb von Microsoft Visual Studio® ein Tool-Window, das in zwei Bereiche unterteilt ist. Auf der linken Seite („Specific Target Area“) werden die Zielsysteme angezeigt. Über die Tabs können Sie zwischen den verschiedenen Extensions umschalten. Auf der rechten Seite („Common Symbol Area“) werden die Details des jeweiligen Zielsystems bzw. des ausgewählten Objekts angezeigt.

Viele Extensions unterstützen eine „Value-Preview“ für Variablen. D. h. wenn Sie eine Variable auswählen und die Leertaste gedrückt halten, wird ein kleines Chart eingeblendet. So können Sie direkt feststellen, ob Daten ankommen oder in einer Datenbank vorhanden sind. Eine Suchleiste über dem rechten Bereich ermöglicht das Filtern / die Reduktion der sichtbaren Symbole. Eine Eingabe müssen Sie mit der [Enter]-Taste bestätigen.

Außerdem zeigt eine Breadcrumb-Navigation-Bar wo Sie sich aktuell befinden.

Extensions

Die nachfolgende Tabelle zeigt eine Übersicht der aktuellen Extensions und von welchem Produkt sie installiert werden. Weitere Informationen zu den Extensions finden Sie in den zugehörigen Dokumentationsabschnitten.

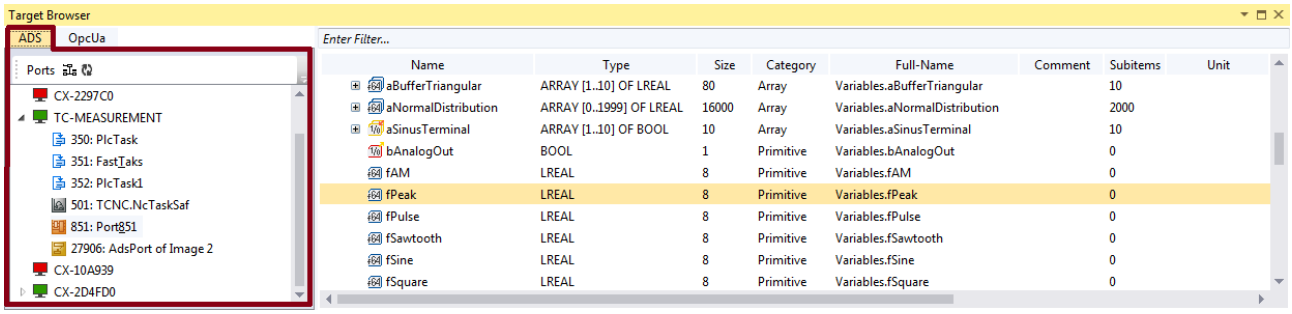
Extension-Name	Produkte
ADS [▶ 54]	TwinCAT XAE, TwinCAT Scope, TwinCAT IoT Data Agent, TwinCAT Database Server, TwinCAT Analytics Service Tool
OPC UA [▶ 56]	TwinCAT Scope, TwinCAT IoT Data Agent
TcAnalytics [▶ 58]	TwinCAT Analytics Workbench und Service Tool, TwinCAT Scope
TcAnalytics File [▶ 60]	TwinCAT Analytics Workbench und Service Tool, TwinCAT Scope
TcDBSrv [▶ 61]	TwinCAT Scope

5.1.1.1.9.1 Extension – ADS

Die Extension TwinCAT Target Browser ADS wird innerhalb des TwinCAT-Systems am häufigsten verwendet.

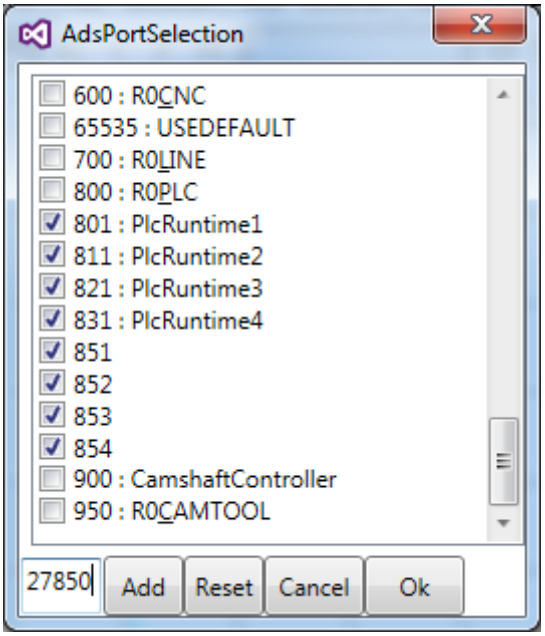
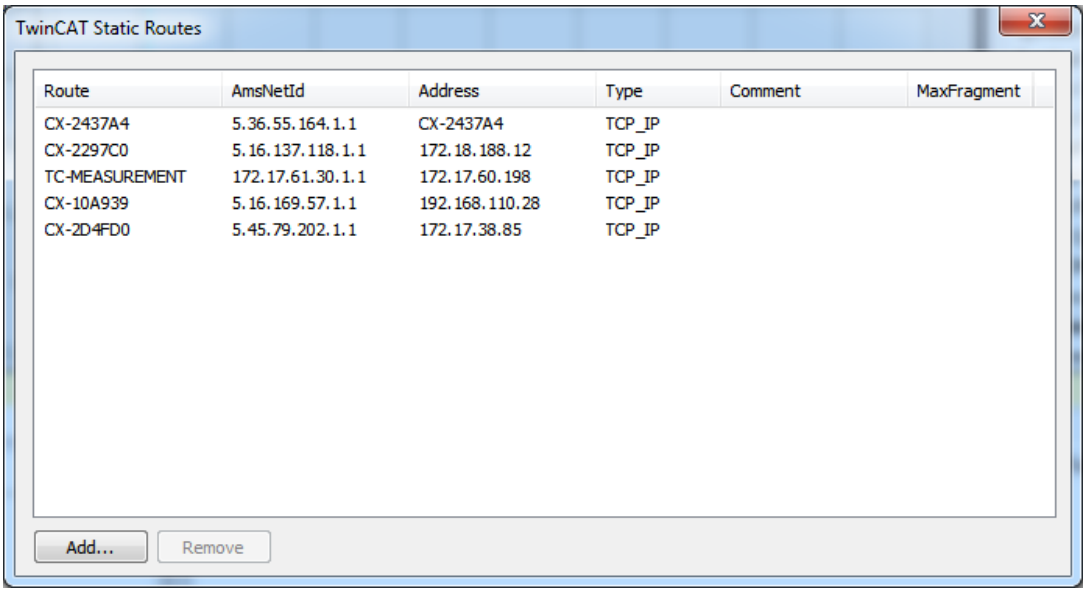
Specific Target Area

Im linken Bereich des TwinCAT Target Browser (ADS) werden alle zum lokalen TwinCAT 3 Engineering eingetragene Zielsysteme in einer Baumstruktur angezeigt. An erster Stelle steht das lokale System, darunter folgen in der Reihenfolge der Eintragung die Zielsysteme wie Industrie-PCs oder Embedded-PCs. Das vorangestellte Bildschirmsymbol zeigt den Zustand des Systems an (grün: Run-Modus, blau: Config-Modus, rot: Stop-Modus bzw. nicht erreichbar). Unterhalb eines Zielsystems werden die zur Verfügung stehenden Default-ADS-Ports aufgelistet. Wenn Sie einen Port auswählen, werden die verfügbaren Symbole/Variablen in der Common Symbol Area auf der rechten Seite des TwinCAT Target Browser angezeigt.



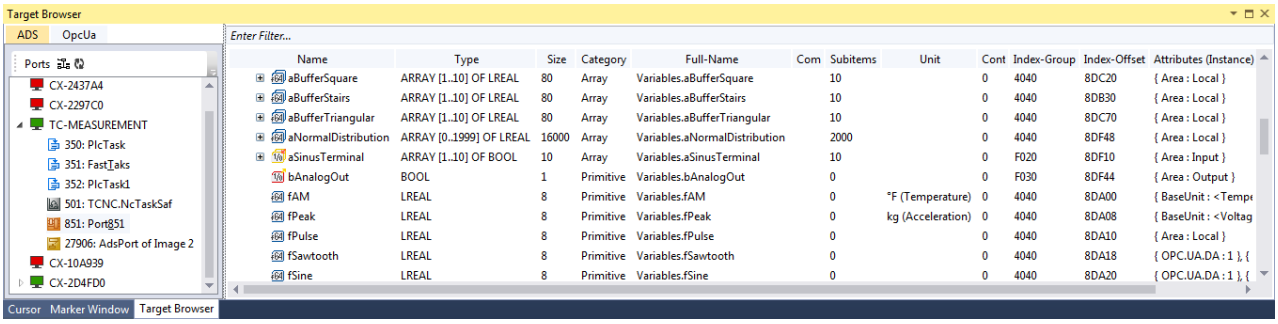
Symbolleiste

Die Symbolleiste der ADS-Extension stellt folgende Funktionen zur Verfügung:

Ports Ports	Wenn ein ADS-Port standardmäßig nicht angezeigt wird, können über diesen Befehl in einer Auswahl weitere Ports hinzugefügt werden. 
Edit Routes	Wenn eine ADS-Route zu einem Zielsystem fehlt, können über diese Schaltfläche weitere Zielsysteme hinzugefügt werden. 
Refresh	Mit dieser Schaltfläche kann die Anzeige der Zielsystem-Zustände manuell aktualisiert werden.

Common Symbol Area

Im rechten Bereich des TwinCAT Target Browsers werden die ADS-Symbole, die am gewählten Port zur Verfügung stehen, angezeigt. Neben dem Namen, dem Datentyp, der Größe und dem Symbolnamen werden beispielsweise auch die Adressen und die Attribute dargestellt. Spezielle Attribute, wie die der Units, werden interpretiert und in eigenen Spalten ausgegeben.

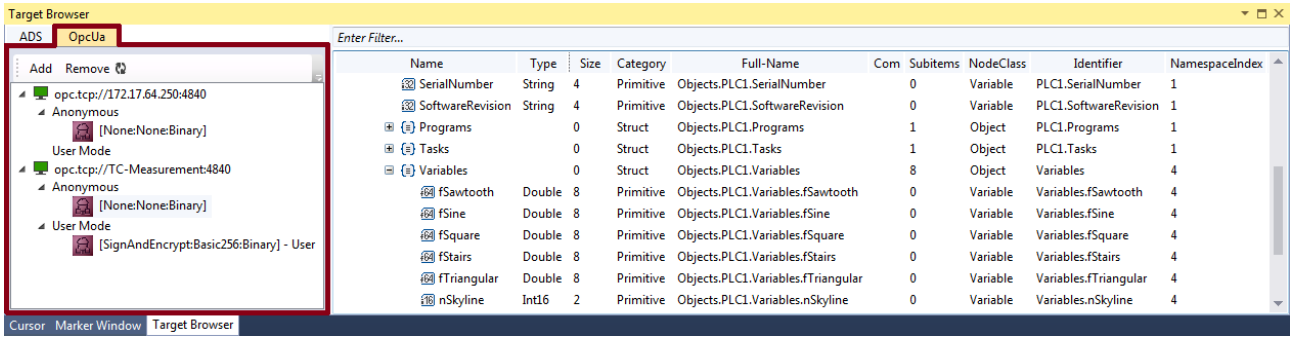


5.1.1.1.9.2 Extension - OPC UA

Die Extension TwinCAT Target Browser OPC UA bildet einen standardisierten Weg in das TwinCAT 3 Engineering hinein.

Specific Target Area

Im linken Bereich des TwinCAT Target Browsers (OpcUa) werden alle OPC UA Server in einer Baumstruktur angezeigt, die über den Symbolleisten-Befehl **Add** hinzugefügt wurden. Das Bildschirmsymbol vor der Server-Bezeichnung auf der ersten Ebene der Baumstruktur zeigt den Verbindungsstatus an. Unterhalb der Server sind die angelegten Endpunkte in „Anonym“ (Anonymous) und „Authentifiziert“ (User Mode) unterteilt. Für jeden Endpunkt wird in Klammern die Art der Verschlüsselung angezeigt. Wenn Sie einen Endpunkt auswählen, werden die verfügbaren OPC UA Nodes auf der rechten Seite des TwinCAT Target Browsers angezeigt (siehe auch: OPC UA Nodes anzeigen [▶ 58])



Symbolleiste

Die Symbolleiste der OPC-UA-Extension stellt folgende Funktionen zur Verfügung:

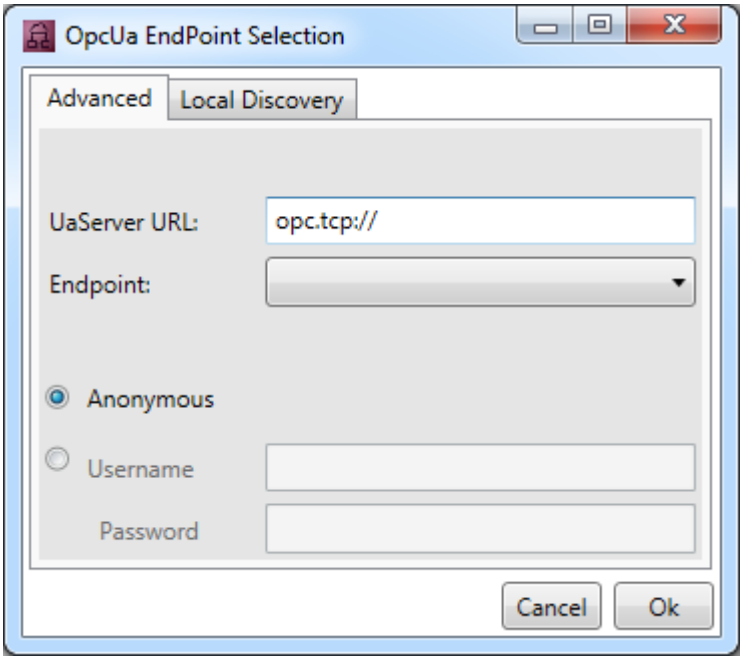
Add 	Mit diesem Befehl können neue Verbindungen zu existierenden OPC UA Servern hergestellt werden (siehe auch: OPC UA Server hinzufügen [▶ 57])
Remove 	Mit diesem Befehl kann ein eingetragener Server entfernt werden.
Refresh 	Mit dieser Schaltfläche kann die Anzeige im Target-Browser-Baum manuell aktualisiert werden.

Common Symbol Area

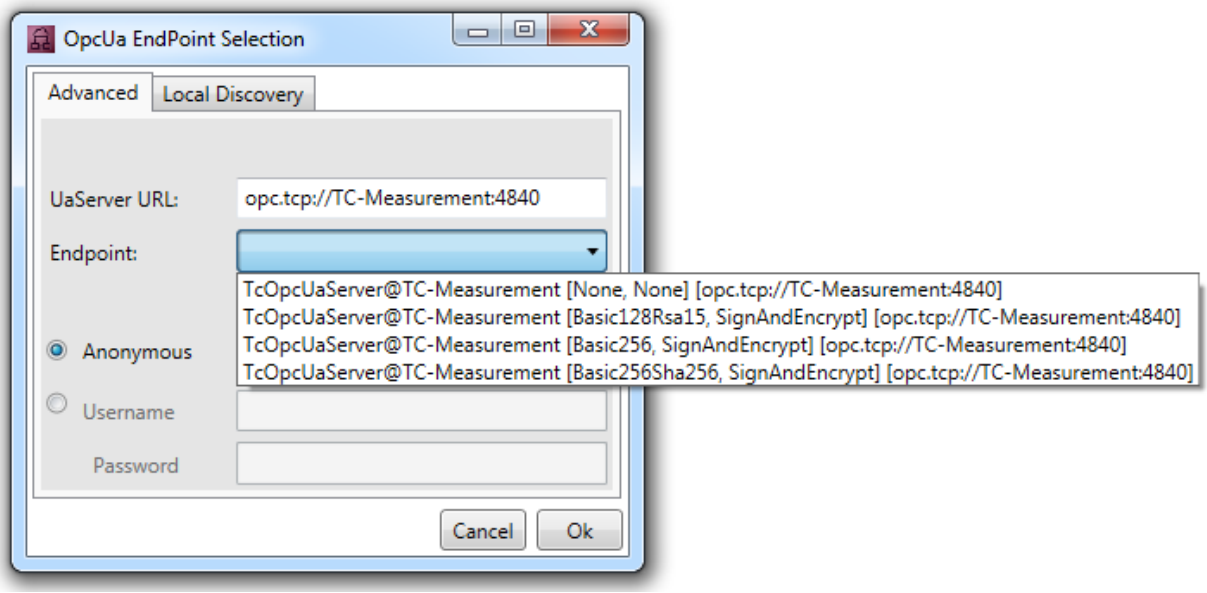
Auf der rechten Seite des TwinCAT Target Browsers werden die verfügbaren OPC UA Nodes angezeigt. Diese spiegeln den hierarchischen Aufbau des SPS-Projekts wider. Neben dem Namen, dem Datentyp, der Größe und dem vollständigen Objektnamen werden beispielsweise auch die Node-Klasse und die Bezeichner dargestellt.

OPC UA Server hinzufügen

1. Klicken Sie in der OPC-UA-Symbolleiste auf **Add**.
   ⇨ Der Dialog **OpcUa EndPoint Selection** öffnet sich.



2. Tragen Sie die URL des Servers ein.
3. Wählen Sie die Endpunkte aus der Drop-Down-Liste aus. Über den entsprechenden Endpunkt können Sie bei OPC UA bestimmen, ob eine und welche Verschlüsselungsart für die Verbindung genutzt wird. Es ist auch möglich zu einem Server mehrere Endpunkte hinzuzufügen. Führen Sie den Befehl **Add** dafür erneut aus.

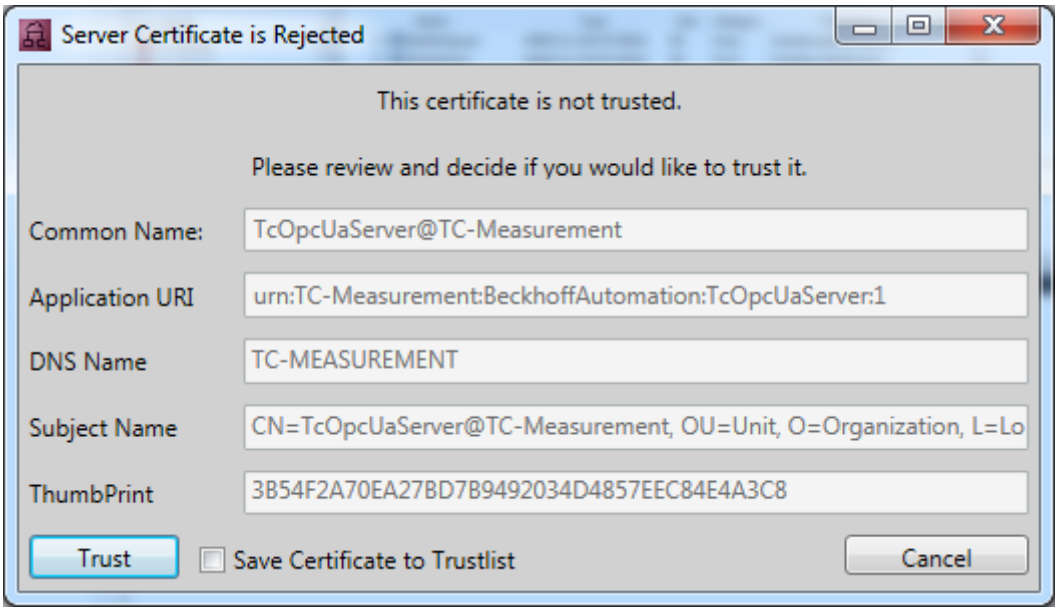


4. Wählen Sie aus, ob es sich um einen anonymen oder authentifizierten Zugriff handelt. Wenn es sich um einen authentifizierten Zugriff handelt, geben Sie einen Benutzernamen (Username) und ein Passwort an. Der authentifizierte Zugriff kann dann erforderlich sein, wenn für den OPC UA Server eine passwortgeschützte Benutzerverwaltung eingerichtet ist (z. B. verschiedene Benutzerkonten mit unterschiedlichen Rechten).
5. Bestätigen Sie den Dialog.

⇨ Der OPC UA Server wird der Baumstruktur im Target Browser mit den ausgewählten Endpunkten hinzugefügt.

OPC UA Nodes anzeigen

Um die verfügbaren OPC UA Nodes anzuzeigen, wählen Sie in der Baumstruktur auf der linken Seite den entsprechenden Endpunkt aus. Wenn Sie einen Endpunkt ohne zertifizierten Zugang auswählen, werden die Nodes direkt angezeigt. Wenn der ausgewählte Endpunkt zertifiziert ist, müssen Sie zunächst in einem entsprechenden Dialog dem Server-Zertifikat vertrauen.



Sie können dem Zertifikat einmalig vertrauen (bis die Visual-Studio-Instanz geschlossen ist) oder es über das Auswahlkästchen **Save Certificate to Trustlist** in die Liste vertrauenswürdiger Zertifikate aufnehmen.

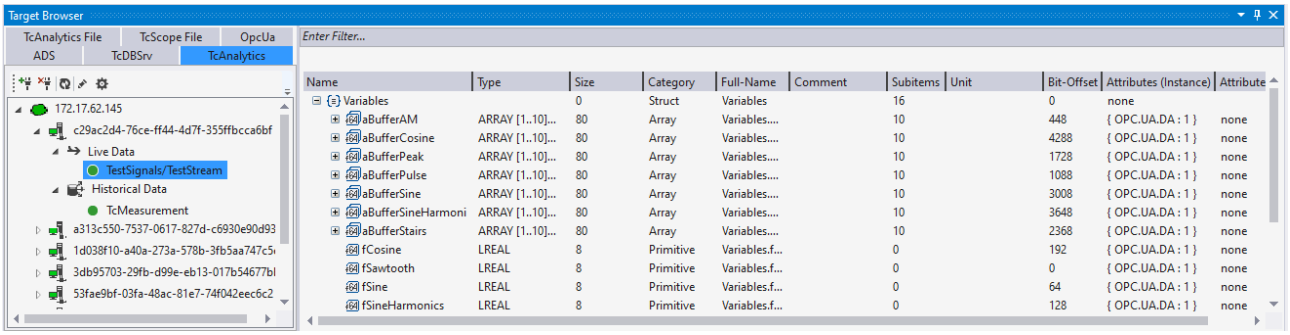
Beim ersten Verbindungsversuch mit einem OPC UA Server ist es zusätzlich erforderlich, dem Zertifikat des Clients (Target Browser) auf der Seite des Servers zu vertrauen. Kopieren Sie dazu das entsprechende Zertifikat im Zertifikatsverzeichnis des OPC UA Servers vom Ordner "rejected" in den Ordner "trusted".

5.1.1.1.9.3 Extension – TcAnalytics

Mit Hilfe der TcAnalytics Extension des TwinCAT Target Browsers können MQTT Datenströme von verschiedenen Brokern und Topics dargestellt werden und für verschiedene Measurement Produkte verfügbar gemacht werden. Hier für müssen nur die gewünschten Symbole der Streams per DragDrop in die entsprechenden Engineering Tools gezogen werden.

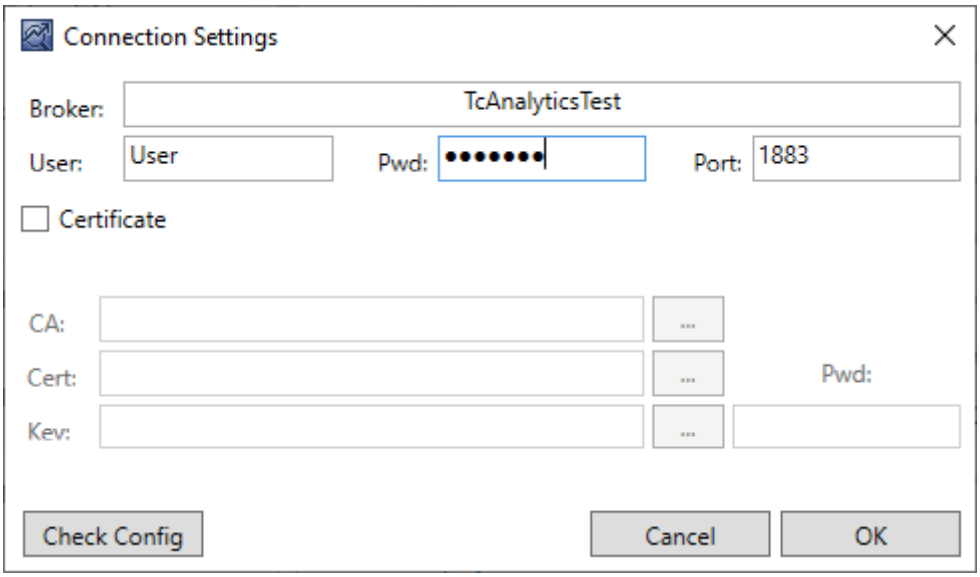
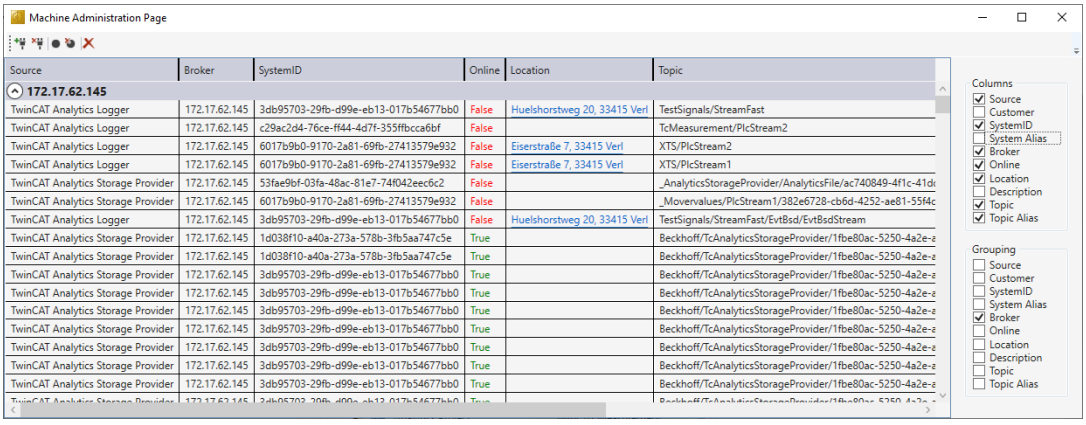
Specific Target Area

Im linken Bereich des TwinCAT Target Browser (TcAnalytics) werden alle Broker und deren Datenströmen in einer Baumstruktur angezeigt. Zusätzlich werden auch historisierte Datenströme angezeigt. An den vorangestellten Symbolen wird der aktuelle Zustand des Systems bzw. des Datenstroms angezeigt (grün: Verfügbar, rot: Nicht erreichbar, grau: Unbekanter Status).



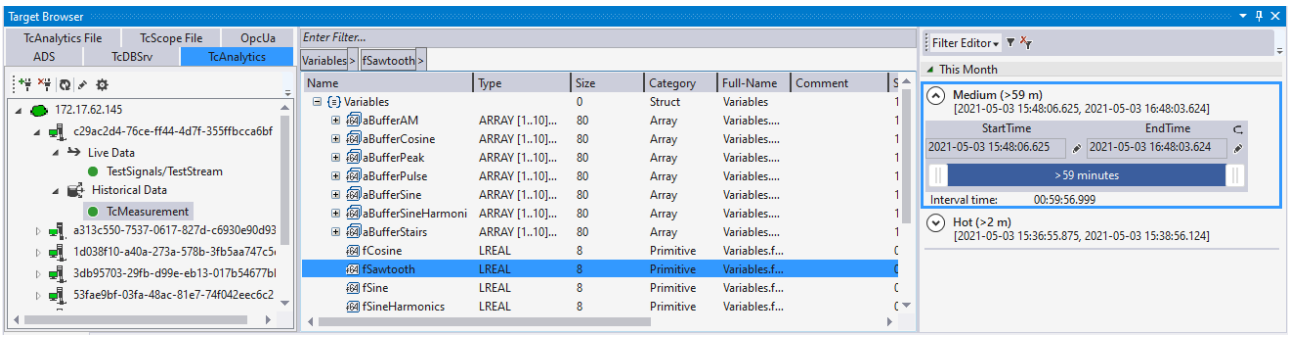
Symbolleiste

Die Symbolleiste der TcAnalytics-Extension stellt folgende Funktionen zur Verfügung:

Broker Verbindung hinzufügen	Über diese Schaltfläche können Broker Verbindungen hinzugefügt werden. 
Broker Verbindung löschen	Mit dieser Schaltfläche können bestehende Verbindungen zu einem Broker aus dem Baum herausgelöscht werden.
Refresh	Mit dieser Schaltfläche kann die Anzeige der Zielsystem-Zustände manuell aktualisiert werden.
Verbindung editieren	Über diese Schaltfläche können die Verbindungsparameter nachträglich geändert werden.
Maschinen Administrations Seite	Über diese Schaltfläche wird die „Maschinen Administrations Seite“ geöffnet. Über diese Seite können die eingehenden Datenströme der verschiedenen „Maschinen“ administriert werden. 

Common Symbol Area

Im rechten Bereich des TwinCAT Target Browsers (TcAnalytics) werden die Symbole der verschiedenen Datenströmen angezeigt. Neben dem Namen, dem Datentyp, der Größe und dem Symbolnamen werden beispielsweise auch Attribute dargestellt. Spezielle Attribute, wie die der Units, werden interpretiert und in eigenen Spalten ausgegeben. Ist eine historisierter Datenstrom ausgewählt, werden zusätzlich noch die einzelnen Aufnahmen und deren Zeitbereiche angezeigt.

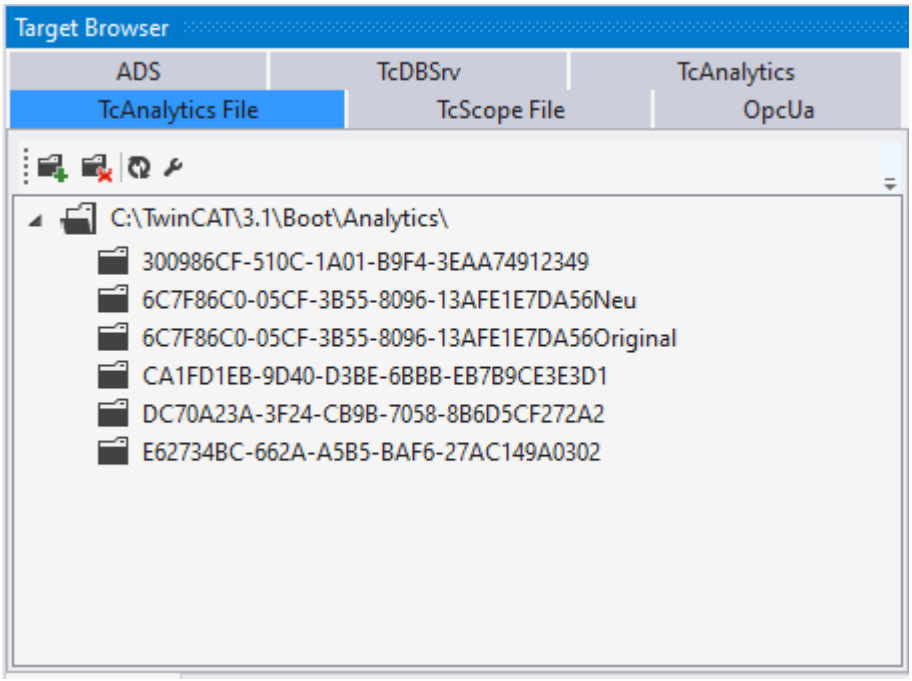


5.1.1.1.9.4 Extension – TcAnalytics File

Mit Hilfe der TcAnalytics Extension des TwinCAT Target Browsers können MQTT Datenströme von verschiedenen Brokern und Topics dargestellt werden und für verschiedene Measurement Produkte verfügbar gemacht werden. Hier für müssen nur die gewünschten Symbole der Streams per DragDrop in die entsprechenden Engineering Tools gezogen werden.

Specific Target Area

Im linken Bereich des TwinCAT Target Browser (TcAnalyticsFile) werden alle Ordner angezeigt, in denen nach AnalyticsFile Ordner gesucht werden soll. Die gefunden AnalyticsFile Ordner werden dann in einer Baumstruktur dargestellt.



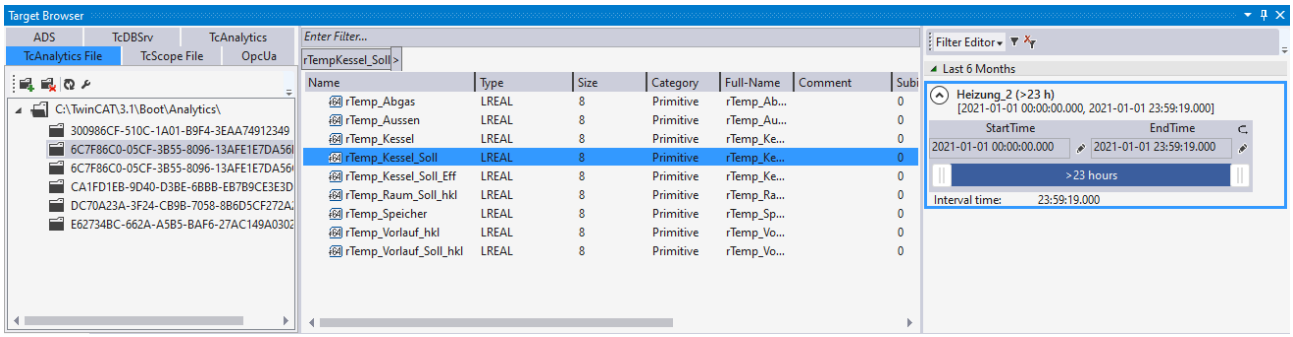
Symbolleiste

Die Symbolleiste der TcAnalyticsFile-Extension stellt folgende Funktionen zur Verfügung:

Ordner hinzufügen	Über diese Schaltfläche können Ordner Pfade hinzugefügt werden in denen nach AnalyticsFile Ordnern gesucht werden soll.
Ordner löschen	Über diese Schaltfläche kann der ausgewählte Ordner aus dem Baum gelöscht werden.
Refresh	Mit dieser Schaltfläche kann die Anzeigen manuell aktualisiert werden.
Eigenschaften	Mit dieser Schaltfläche können verschiedene Eigenschaften der TcAnalyticsFile-Extension angepasst werden.

Common Symbol Area

Im rechten Bereich des TwinCAT Target Browsers (TcAnalyticsFile) werden die Symbole der verschiedenen AnalyticsFiles angezeigt. Neben dem Namen, dem Datentyp , der Größe und dem Symbolnamen werden beispielsweise auch Attribute dargestellt. Spezielle Attribute, wie die der Units, werden interpretiert und in eigenen Spalten ausgegeben. Zusätzlich werden noch die einzelnen Aufnahmen und deren Zeitbereiche angezeigt.

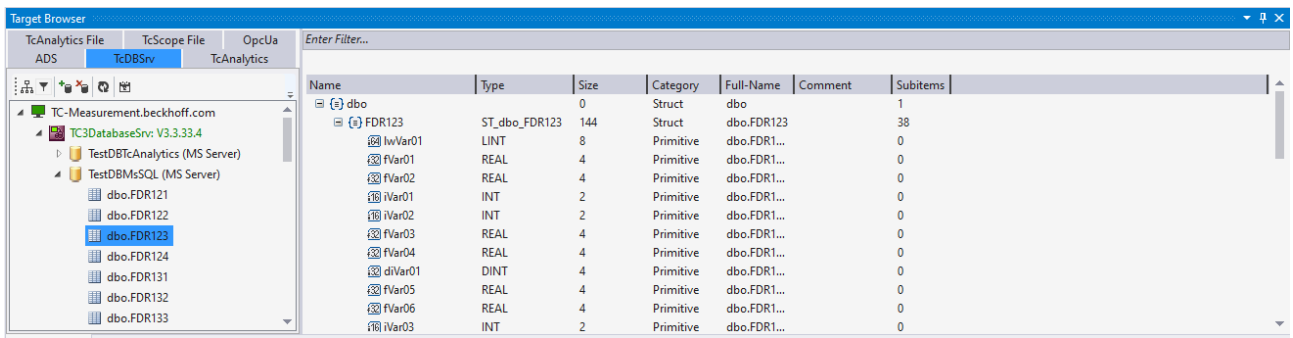


5.1.1.1.9.5 Extension – TcDBSrv

Mit Hilfe der TcDBSrv Extension des TwinCAT Target Browsers können Datensätze aus Datenbanken über den TwinCAT Database Server in TwinCAT Scope dargestellt werden. Hier für müssen nur die gewünschten Spalten der Tabellen per DragDrop ins TwinCAT Scope gezogen werden. Es wird automatisch ein entsprechender SQL Befehl generiert, welcher natürlich noch von Hand individualisiert werden kann.

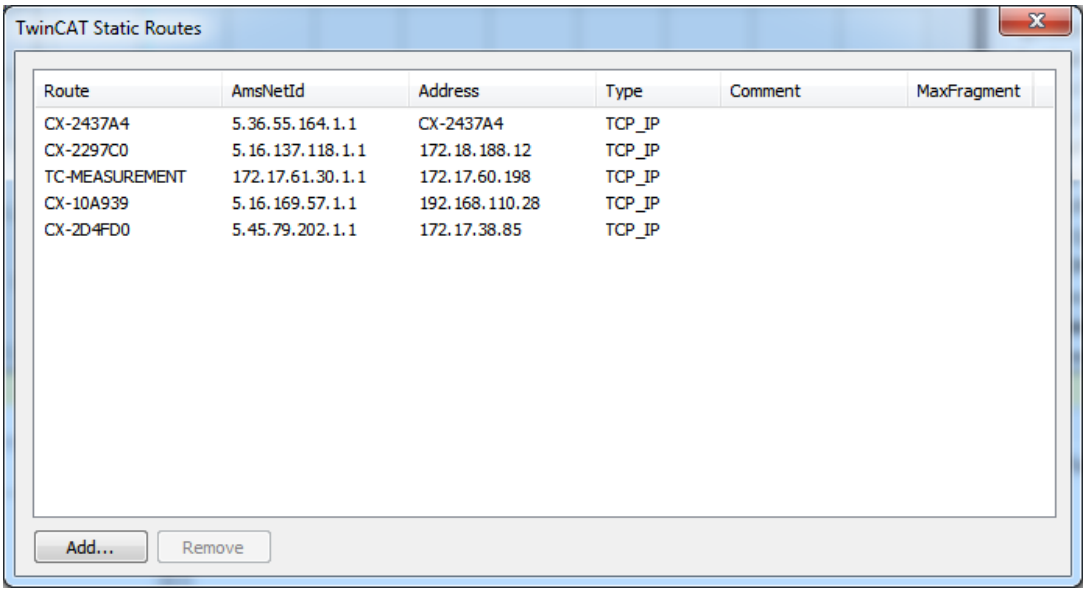
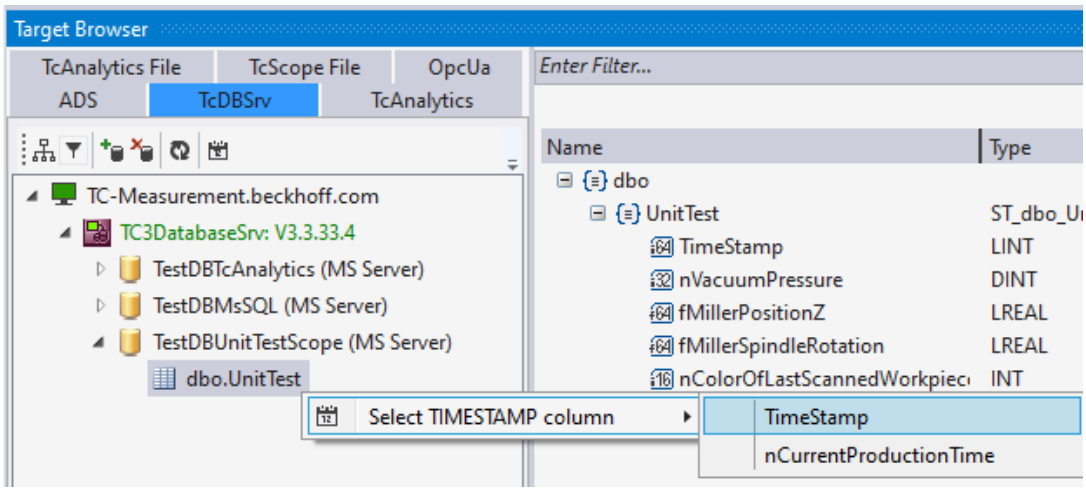
Specific Target Area

Im linken Bereich des TwinCAT Target Browser (TcDBSrv) werden alle zum lokalen TwinCAT 3 Engineering eingetragene Zielsysteme in einer Baumstruktur angezeigt. An erster Stelle steht das lokale System, darunter folgen in der Reihenfolge der Eintragung die Zielsysteme wie Industrie-PCs oder Embedded-PCs. Das vorangestellte Bildschirmsymbol zeigt den Zustand des Systems an (grün: Run-Modus, blau: Config-Modus, rot: Stop-Modus bzw. nicht erreichbar). Unterhalb eines Zielsystems werden die zur Verfügung stehenden TwinCAT Database Server Instanzen aufgelistet. Darunter finden sie alle konfigurierten Datenbanken mit ihren erreichbaren Tabellen. Wird ein Tabellenknoten ausgewählt, so wird in der „Common Symbol Area“ die einzelnen Spalten der Tabelle dargestellt.



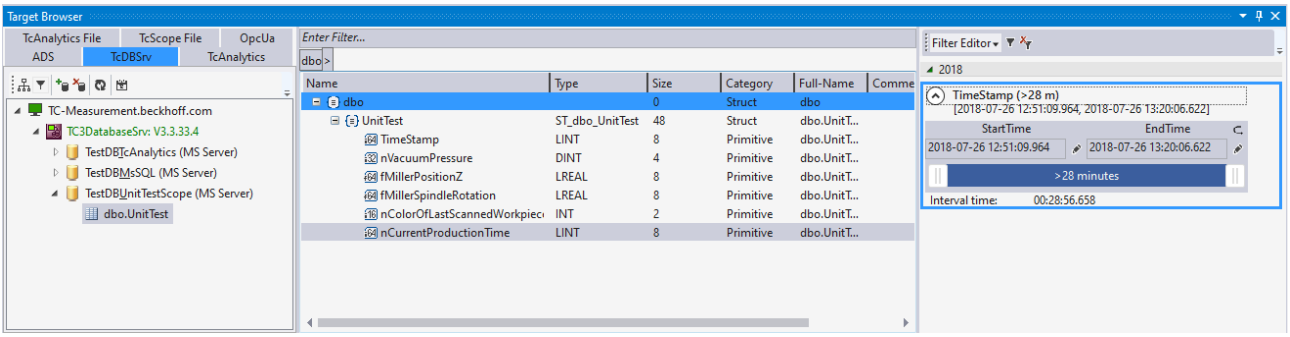
Symbolleiste

Die Symbolleiste der TcDBSrv-Extension stellt folgende Funktionen zur Verfügung:

Edit Routes	Wenn eine ADS-Route zu einem Zielsystem fehlt, können über diese Schaltfläche weitere Zielsysteme hinzugefügt werden. 
Filter	Ist der Filter aktiv werden nur die Routen angezeigt, wo ein TwinCAT Database Server installiert und erreichbar ist.
Datenbank hinzufügen	Über diese Schaltfläche können Datenbankkonfigurationen angelegt werden. Es können Datenbanken aus dem Datenbankpool, oder neue Konfigurationen am TwinCAT Database Server angelegt werden. Es öffnen sich die bekannten Konfigurationseditoren.
Datenbank löschen	Über diese Schaltfläche kann die ausgewählte Datenbankkonfiguration am TwinCAT Database Server gelöscht werden.
Refresh	Mit dieser Schaltfläche kann die Anzeige der Zielsystem-Zustände manuell aktualisiert werden.
Zeitstempelspalte festlegen	Über diese Schaltfläche oder über das Kontextmenü am Tabellenknoten kann eine Spalte als Zeitstempel festgelegt werden und der Zeitbereich der Tabelle wird anhand dieser Spalte ausgelesen. 

Common Symbol Area

Im rechten Bereich des TwinCAT Target Browsers (TcDBSrv) werden die Tabellen mit ihren Spalten angezeigt. Neben dem Namen, dem Datentyp wird auch die Größe der Spalten dargestellt. Ist eine Spalte als Zeitstempelspalte definiert, so wird der Zeitbereich der Daten angezeigt



5.1.1.1.10 Support Information Report

Der Support Information Report ist ein Tool zum Sammeln von Produktinformationen, um mit dem technischen Beckhoff Support in Kontakt zu treten. Durch das Sammeln produktrelevanter Daten wie TwinCAT Version/Build, Produktversion, Image-Version und Gerätetyp wird der Email-Verkehr deutlich reduziert und eine schnellere Beratung ermöglicht.

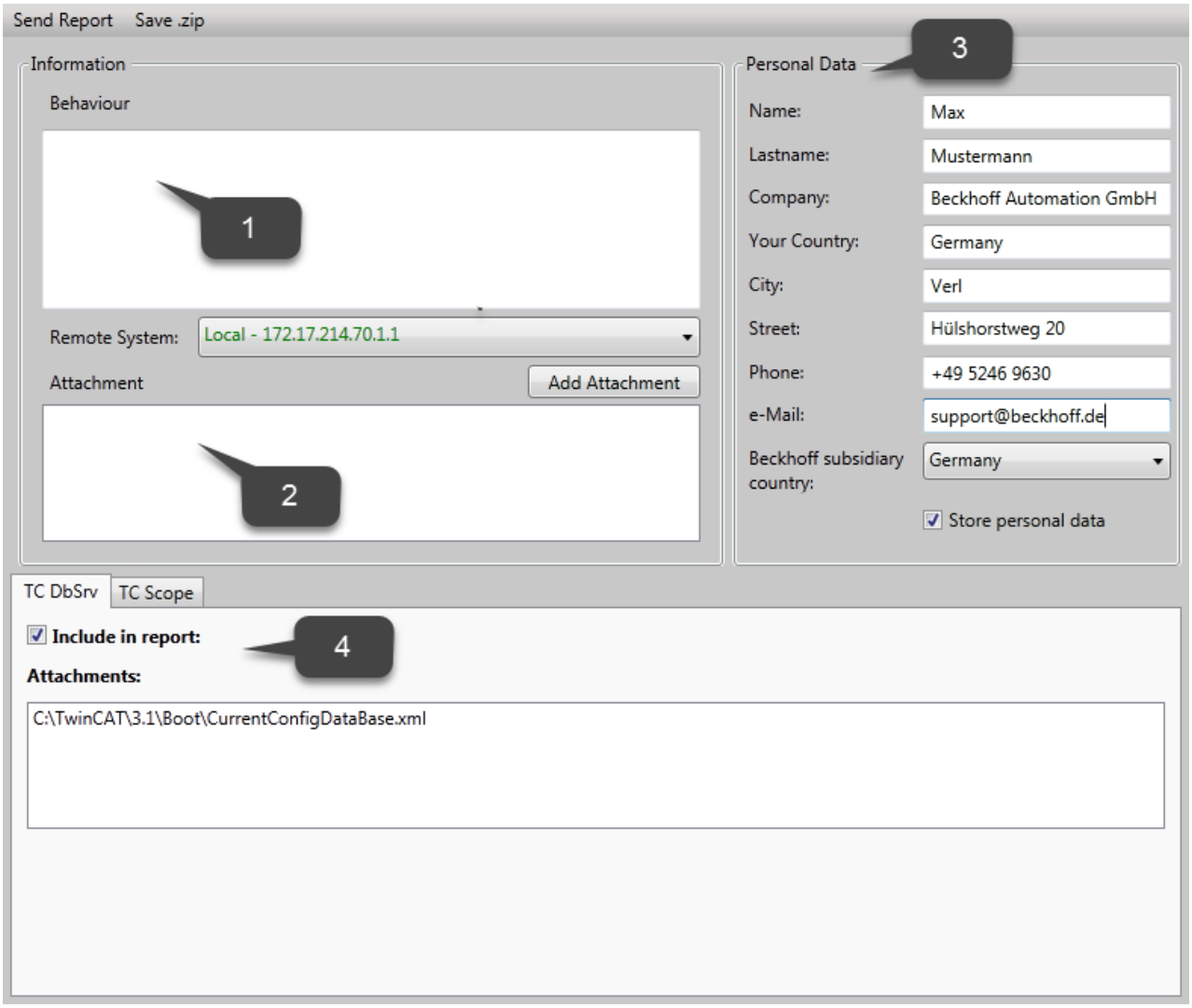
PlugIn-Mechanismus

Verschiedene Beckhoff Produkte integrieren sich in den Support Information Report über einen PlugIn-Mechanismus. Sie finden daher in den spezifischen Produkten wie z. B. dem TwinCAT Database Server im entsprechenden Produktmenü einen Eintrag zum Support Information Report.

Support Information Report erstellen und abschicken

✓ Ein Support Information Report ist geöffnet.

1. Beschreiben Sie das aufgetretene Verhalten im Textfeld **Behavior** möglichst ausführlich.
2. Fügen Sie dem Report im Bereich **Attachment** optional über die Schaltfläche **Add Attachment** zusätzliche Dateien (Screenshots, …) hinzu. Optional können Sie auch Dateien über den Fernzugriff auswählen. Wählen Sie dazu ein Target aus der Dropdown-Liste **Remote System** aus. Je nach ausgewähltem Target ist hier sogar ein Browsen von Windows-CE-Geräten möglich.
3. Geben Sie Ihre Kontaktdaten ein und wählen Sie eine Beckhoff Länder-Niederlassung aus. Diese Angabe ist erforderlich, um den Support Information Report abzuschicken.
4. Optional können Sie Ihre Kontaktdaten für die erneute Verwendung des Support Information Reports speichern. Aktivieren Sie dazu das Auswahlkästchen **Store personal data**.
5. Im unteren Bereich des Support Information Reports finden Sie die produktspezifischen PlugIns. Aktivieren Sie das Auswahlkästchen **Include in report**. Die für das Produkt notwendigen Informationen werden, soweit vorhanden, automatisch hinzugefügt. Im Screenshot ist beispielhaft die aktuelle Konfiguration von einem TwinCAT Database Server als XML-Datei dargestellt.
6. Schicken Sie den Support Information Report ab:
   - Wenn das Gerät eine E-Mail-Anbindung hat, können Sie den Support Information Report direkt über die Schaltfläche **Send Report** an die Beckhoff Länder-Niederlassung schicken.
   - Wenn das Gerät keine E-Mail -Anbindung hat, können Sie den Support Information Report über die Schaltfläche **Save .zip** zunächst lokal als .zip-Datei speichern und anschließend über FTP, USB etc. zugänglich machen und verschicken.



5.1.1.2 Configure Mode

Dieses Kapitel ist eine Zusammenstellung aller nötigen Informationen, um den Konfigurationsmodus (Configure Mode) des TwinCAT Database Servers zu nutzen. Es behandelt folgende Themen:

- Erstellen eines Projektes
- Anlegen und Einstellen einer Datenbankkonfiguration
- Anlegen und Einstellen von Autologgruppen
- Aktivieren eines Database-Server-Projektes
- Überwachen und Steuern des automatischen Loggens

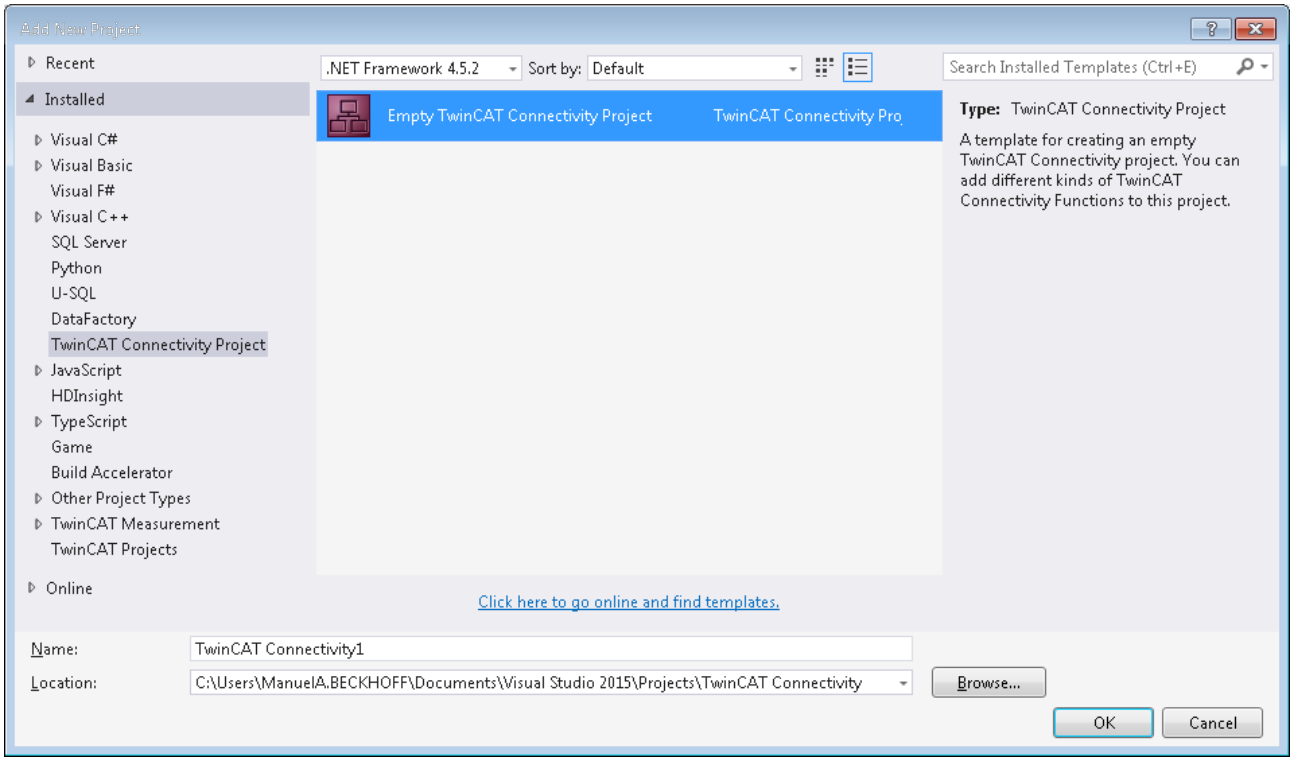
Configure Mode

Im Configure Mode ist der Hauptteil der Arbeit im Konfigurator zu verrichten. Hier muss die Konfiguration für die gewünschte Datenbank und für die AutoLog-Gruppe eingestellt werden. Für die Konfiguration der AutoLog-Gruppe kann der Target Browser genutzt werden, um online auf ein Zielsystem zuzugreifen und die zu kommunizierenden Variablen auszuwählen. Nutzt der Anwender die Option **AutoStart**, so wird die Kommunikation mit der konfigurierten Datenbank direkt mit dem Aufstarten des TwinCAT-Systems aufgenommen. Wird die Option **Manual** gewählt, muss die Kommunikation durch den Funktionsbaustein FB_PLCDBAutoLog [▶ 164] oder für den Autolog View angesteuert werden.

Projekt erstellen

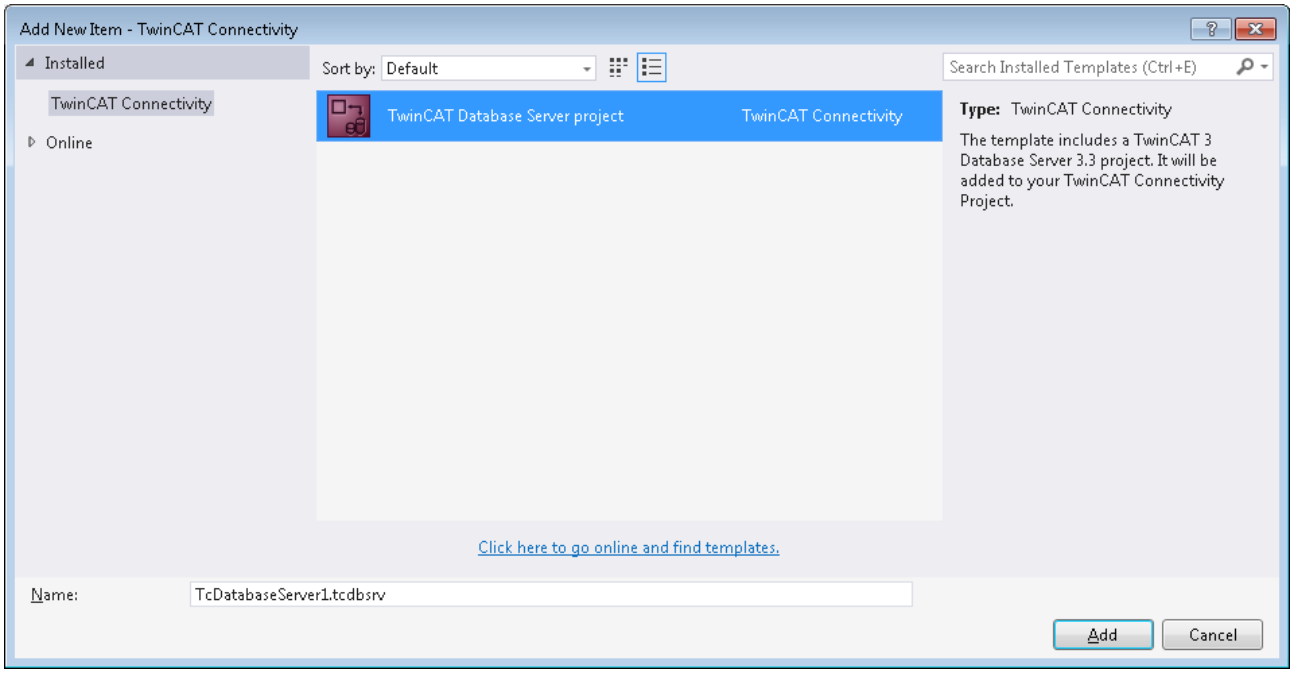
Durch die TwinCAT-Connectivity-Erweiterung für Visual Studio steht eine neue Projektvorlage zur Verfügung. Beim Erstellen eines neuen Projektes erscheint nun die Kategorie **TwinCAT Connectivity Project** in der Auswahl.

Um ein neues TwinCAT-Connectivity-Projekt zu erstellen, wählen Sie das **Empty TwinCAT Connectivity Project**, legen den Projektnamen und den Speicherort fest und fügen es mit **OK** der Solution hinzu. TwinCAT-Connectivity-Projekte bzw. TwinCAT-Database-Server-Projekte können so komfortabel neben TwinCAT- oder anderen Visual-Studio-Projekten angelegt werden.

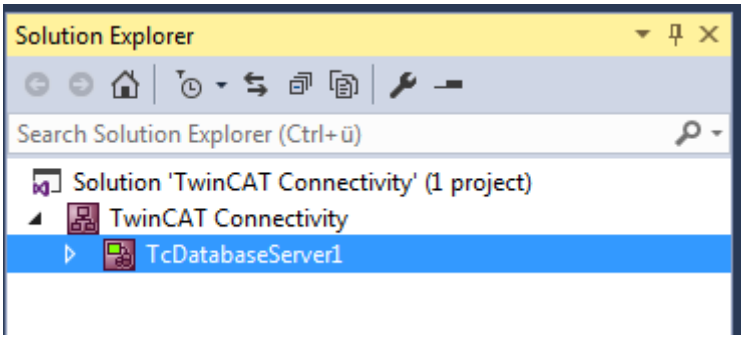


In der Solution erscheint ein neuer Projektknoten. Unterhalb des Connectivity-Projektknotens können Sie Subprojekte der unterstützten Connectivity-Funktionen ergänzen.

Mit **Add** können Sie dem TwinCAT-Connectivity-Projekt ein neues TwinCAT-Database-Server-Projekt hinzufügen. Das TwinCAT-Database-Server-Projekt befindet sich in der Auflistung der vorhandenen Item Templates.



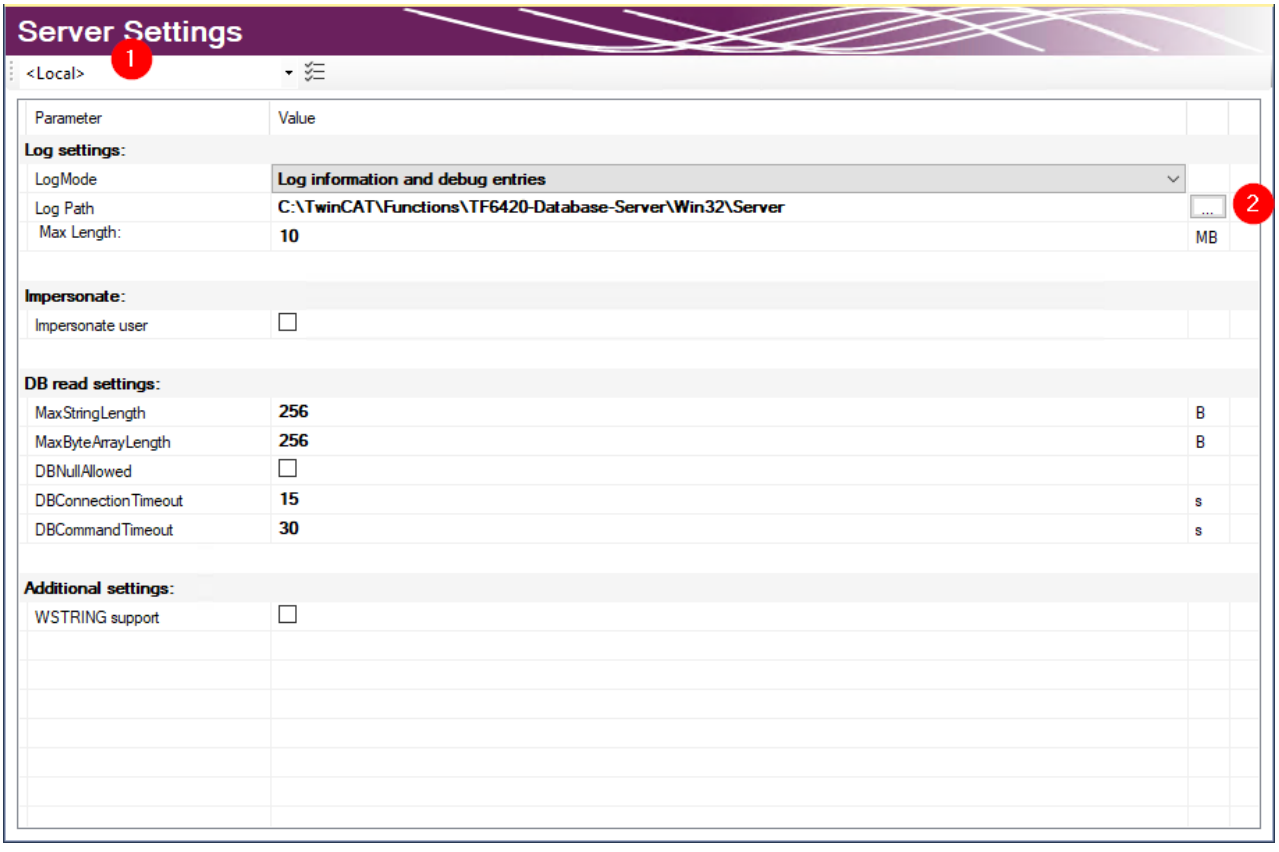
Unterhalb des TwinCAT-Connectivity-Knotens wird ein neues TwinCAT-Database-Server-Projekt angelegt.



Dieses dient nun als Basis für die anstehende Konfiguration eines TwinCAT Database Servers. Das Dokument können Sie sowohl über die Eigenschaften im Eigenschaftsfenster, als auch über einen Editor bearbeiten.

Einem Connectivity-Projekt können Sie beliebig viele TwinCAT-Database-Server-Projekte oder andere Projekte hinzufügen, und damit auch mehrere Konfigurationen in einem Connectivity-Projekt einstellen.

Editor für Server-Einstellungen



Im Editor **Server Settings** können Sie die Einstellungen für den TwinCAT Database Server bearbeiten. Diese Einstellungen betreffen den entsprechenden Server im Allgemeinen. Im Drop-down-Menü (1) können Sie das Zielsystem über die Ams-NetId angegeben. Dafür müssen Sie über TwinCAT eine Route zum Zielsystem anlegen. Wird eine fertige Konfiguration übertragen, werden die Einstellungen auf den TwinCAT Database Server dieses Zielsystems gespeichert.

In den **Log settings** konfigurieren Sie Einstellungen zum Aufzeichnen von Fehlerfällen. In einem Fehlerfall erzeugt der Database Server einen detaillierten Eintrag in einer Logdatei. Diese können Sie mit dem Information Log Viewer [▶ 51] auslesen. In den **Log Settings** geben Sie einen Pfad zum Ablageort und die

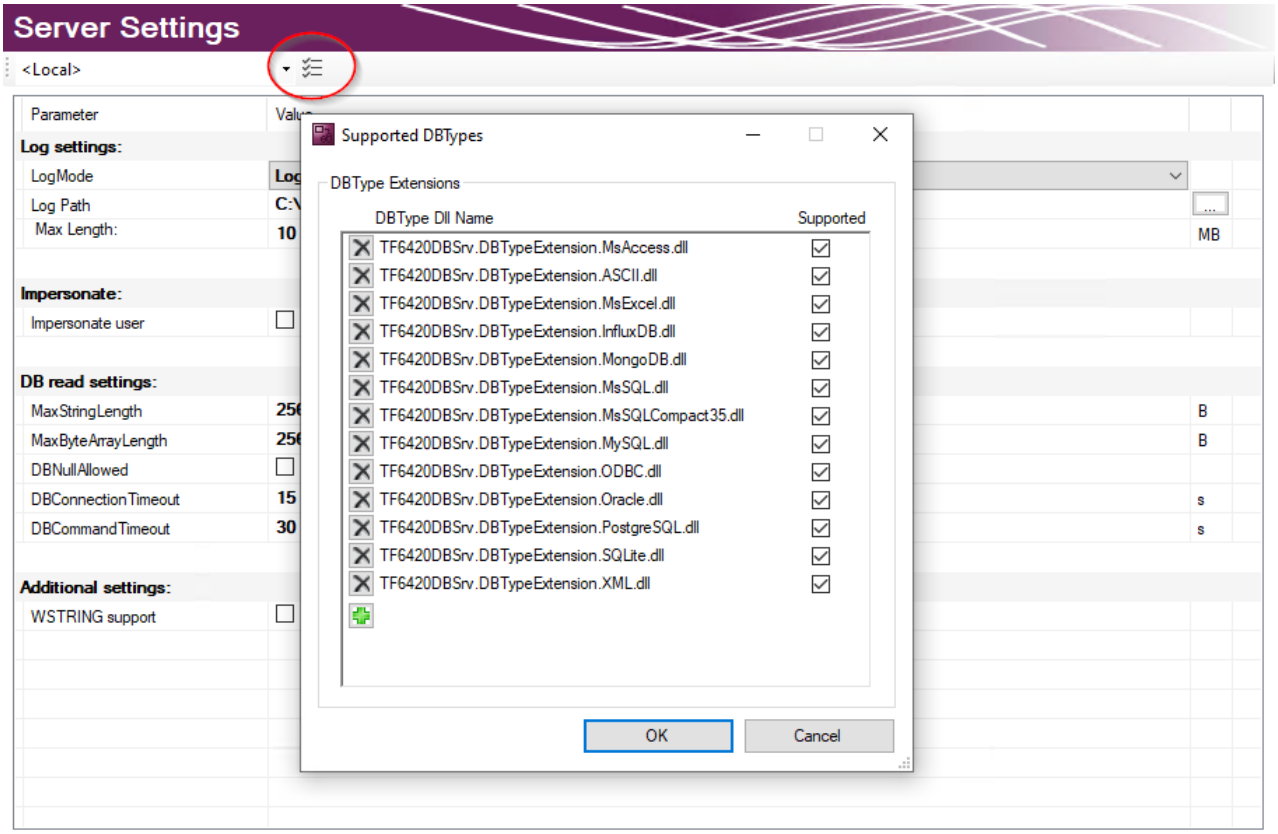
maximale Größe der Datei an. Zusätzlich können Sie die Genauigkeit des Logs beeinflussen. Wir empfehlen, aus Performancegründen, das Loggen nach erfolgter Fehleranalyse wieder abzuschalten, wenn es nicht mehr benötigt wird.

Bei Netzwerkzugriff auf dateibasierende Datenbanken wie Access Datenbanken oder SQL Compact Datenbanken müssen Sie die Option **Impersonate** setzen, damit sich der TwinCAT Database Server mit dem Netzwerklaufwerk verbinden kann. **Diese Funktion wird zurzeit nicht unter Windows CE unterstützt.**

Sie können weitere Einstellungen konfigurieren, um das Lesen aus der Datenbank zu steuern. Diese Einstellungen beziehen sich auf den TwinCAT Database Server auf dem Zielsystem:

MaxStringLength	Maximale String-Länge der Variablen in der SPS
MaxByteArrayLength	Maximale Byte-Array-Länge der Variablen in der SPS
DBNullAllowed	Gibt an, ob NULL-Werte im TwinCAT Database Server angenommen werden.
DBConnectionTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem Verbindungsaufbau von einem Verbindungsfehler ausgeht.
DBCommandTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem abgesendeten Kommando von einem Verbindungsfehler ausgeht. Bei hohen Datenmengen kann, je nach Datenbank und Infrastruktur ein Befehl durchaus sehr viel Zeit in Anspruch nehmen.

Unterstützte Datenbanktypen

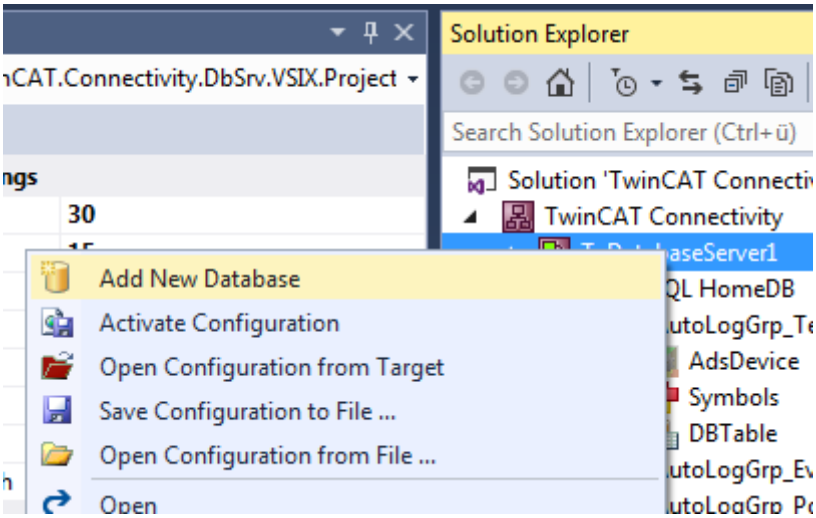


Die installierten Datenbanktypen können in den Server Einstellungen ausgewählt werden. Standardmäßig sind alle installierten Datenbanken angewählt. Der TwinCAT 3 Database Server wird die entsprechenden Datenbankschnittstellen laden. Auf diese Weise können nicht genutzte Datenbanken auf dem Zielsystem abgewählt werden.

Eine neue Datenbankkonfiguration hinzufügen

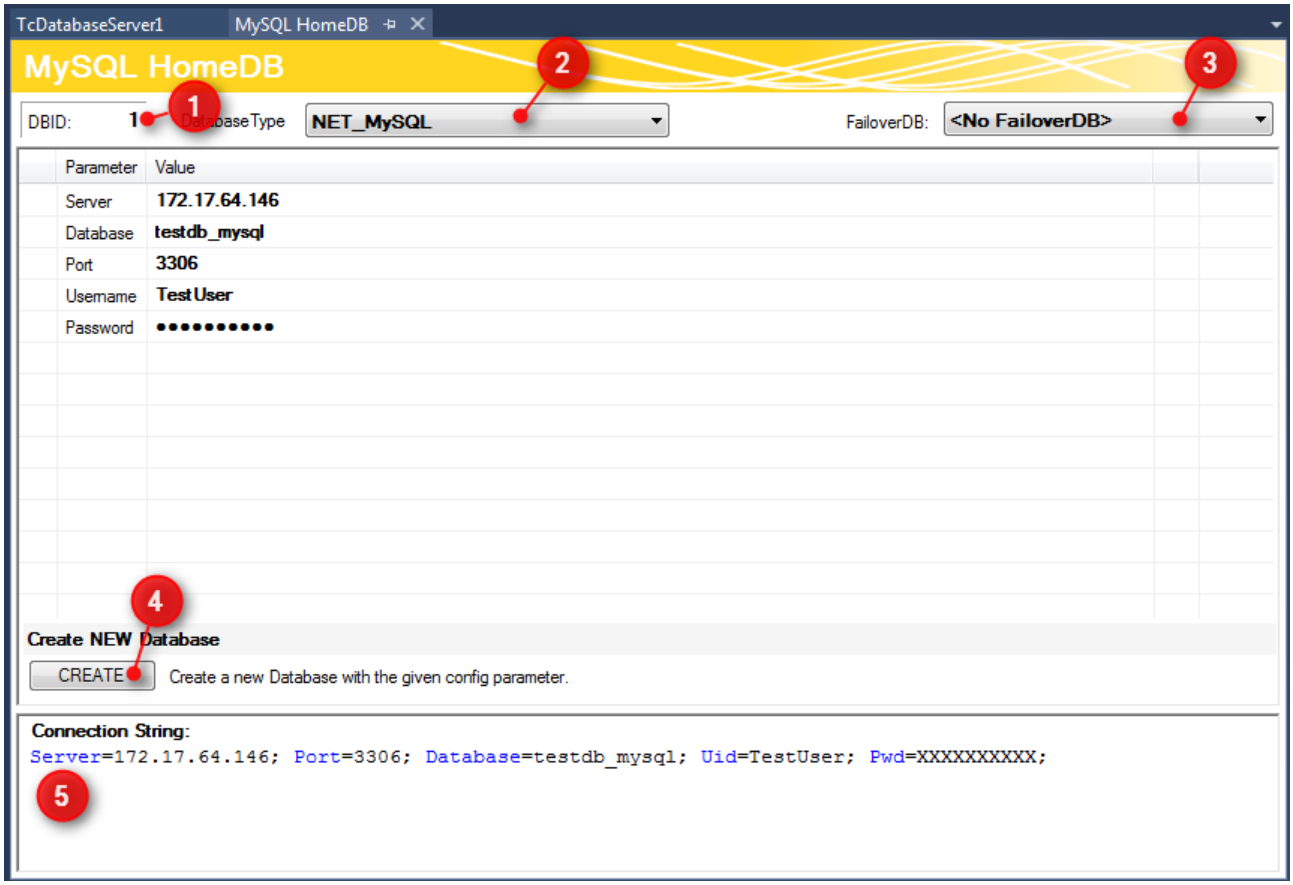
Die Datenbankkonfiguration wird benötigt, um dem Database Server alle nötigen Informationen zur Datenbankverbindung mitzuteilen.

Eine neue Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new Database** über das Kontextmenü eines Database-Server-Projektes oder das entsprechende Kommando in der Toolbar hinzufügen.



Eine neue Datenbankkonfiguration wird als Datei auf im Projektordner hinzugefügt und in das Projekt eingebunden. Wie bei allen Visual-Studio-Projekten, werden die Informationen über die neuen Dateien im Connectivity-Projekt hinterlegt.

Editor für Datenbankkonfigurationen

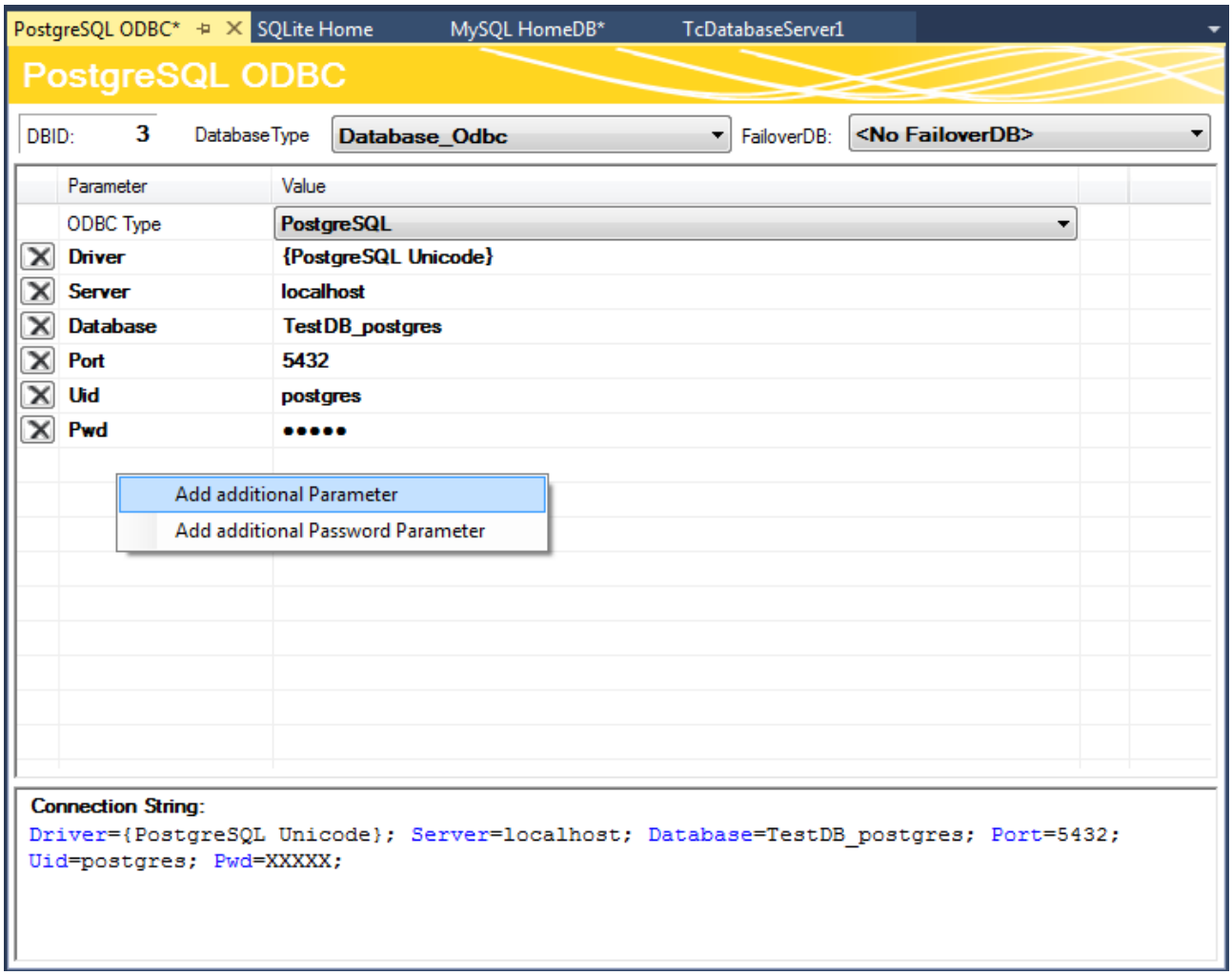


Im oberen Teil des Editors wird die Datenbank ID angezeigt (1), die für einige Funktionsbausteine in der SPS benötigt wird. Den Datenbanktyp der Zieldatenbank können Sie aus dem Drop-down-Menü auswählen (2). Hier können Sie auch die Odbc-Schnittstelle für eine Datenbank nutzen, die noch nicht unterstützt wird. Beachten Sie, dass je nach Datenbank nicht alle Funktionen des TwinCAT Database Servers gewährleistet werden.

Außerdem können Sie optional eine sogenannte FailOver-Datenbank (3) auswählen, welche im ‚Configure'-Mode beim Fehlerfall einspringt. Bei einem Verbindungsabbruch zum Netzwerk kann in einem solchen Fall automatisch gewährleistet werden, dass keine Daten verloren gehen und an anderer Stelle gespeichert werden.

Für jede Datenbank [▶ 126] stehen zusätzlich weitere einstellbare Parameter zur Verfügung. Je nach Datenbank wird ein Connection String (5) erstellt, welcher die Verbindung zur Datenbank beschreibt. Diese Anzeige dient zur Transparenz Ihrer eingestellten Parameter.

Über die Schaltfläche **CREATE** (4) können Sie eine neue Datenbank erstellen. Diese Funktion wird nur angezeigt, wenn sie von der jeweiligen Datenbank unterstützt wird.



Sie können auch unbekannte Datenbanken mit einer ODBC-Schnittstelle konfigurieren. Dafür wählen Sie im in der Drop-down-Liste **ODBC Type** den Eintrag „Unknown Database" und fügen über die Befehle im Kontextmenü Parameter hinzu. Diese können auch Passwörter beinhalten, welche dann verschlüsselt abgespeichert werden. Daraus kann der gewünschte Connection String zusammengestellt werden. Beachten Sie, dass nur begrenzte Funktionen des TwinCAT Database Servers genutzt werden können. Nur die expliziten Funktionsbausteine des SQL Expert Modes werden unterstützt.

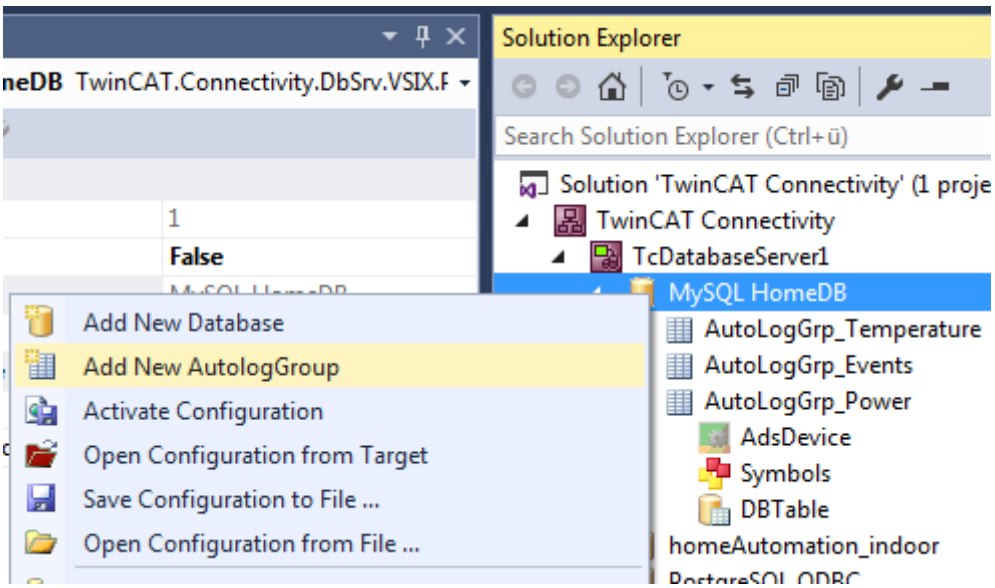
Failover Datenbank

Der TwinCAT 3 Database Server verfügt über die Funktion von Failover Datenbanken. Diese Funktion bietet die Möglichkeit, um bei Verbindungsverlust oder anderen Problemen mit der eingerichteten Datenbank auf eine andere Datenbank umzusteigen, um möglichen Datenverlust zu vermeiden. Diese Funktion wird nur vom ‚Configure Mode' unterstützt. Beim automatischen Wegschreiben, wird im Fehlerfall die entsprechend andere Datenbank genutzt. Die Tabelle der ersten Datenbank muss dafür der zweiten gleichen.

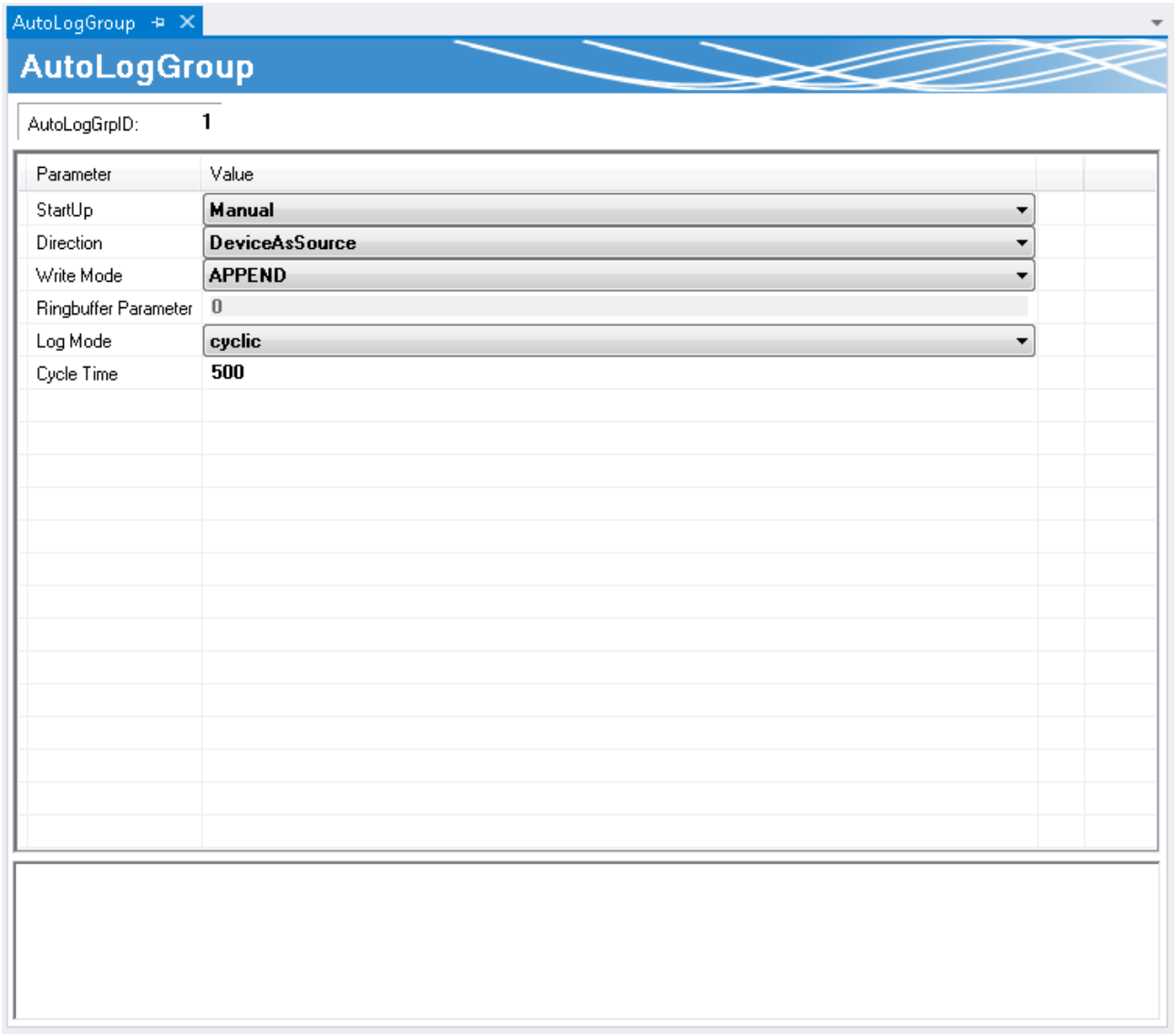
Eine neue Autologgruppe hinzufügen

In den Autologgruppen befinden sich die Informationen, welche Variablen der SPS mit welchen Variablen aus den Datenbanken synchronisiert werden sollen. Zusätzlich werden hier Informationen über die Synchronisationszeitpunkte und der Art der Synchronisation hinterlegt.

Eine neue AutoLog-Gruppe für die Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new AutologGroup** im Kontextmenü einer Datenbankkonfiguration oder über die Toolbar hinzufügen. Diese AutoLog-Gruppen beziehen sich auf die übergeordnete Datenbank.



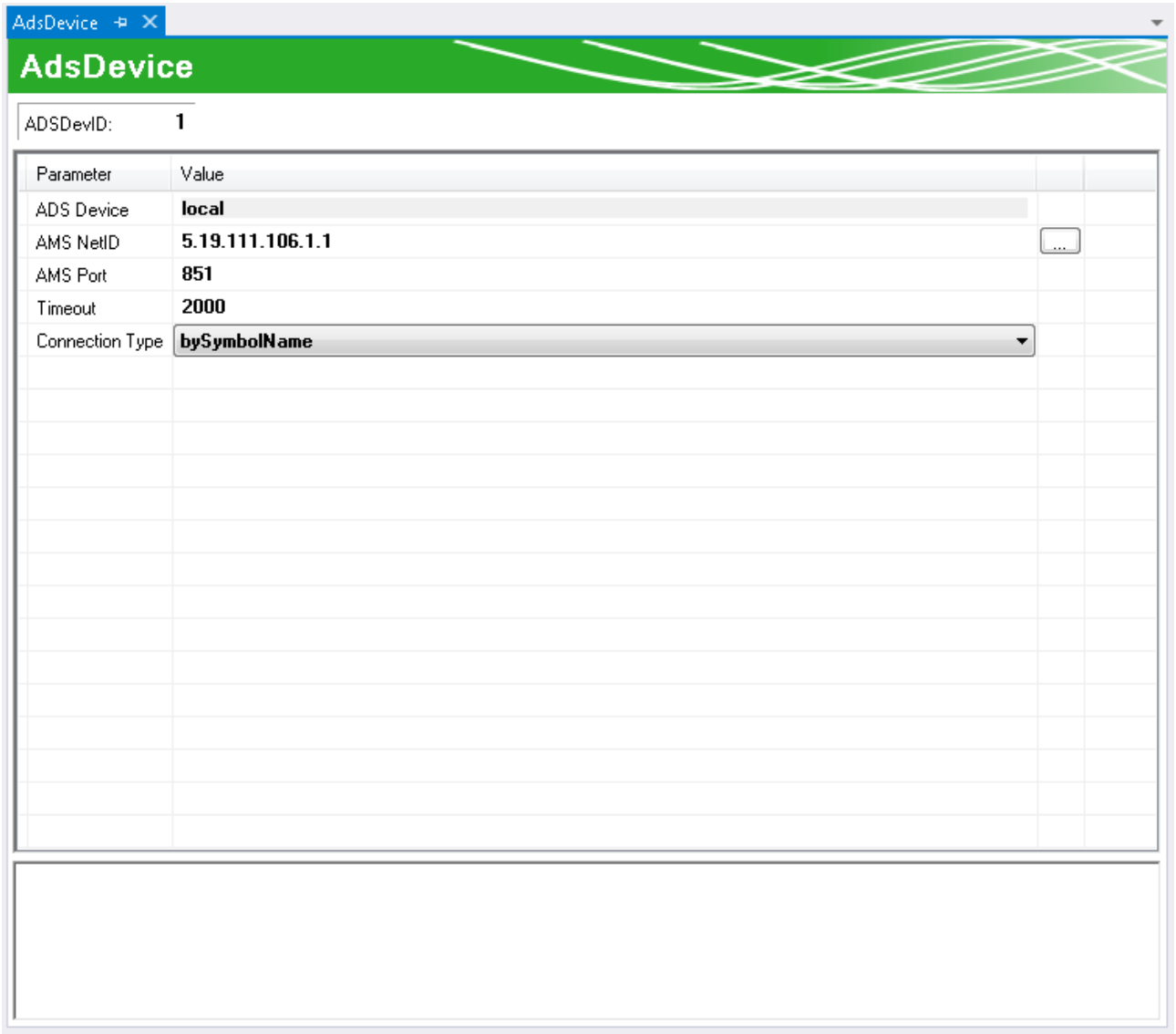
Eine neue AutoLog-Gruppe sowie die dazugehörigen Komponenten werden als Dateien im Projektordner hinzugefügt und im Projekt eingebunden. Dazu gehören das Ads Device, die Symbolgruppen und die Tabelleneinstellungen. Um diese Dateien im Projekt zu speichern, sollten Sie die TwinCAT-Connectivity-Projektdatei speichern. Diese Dateien können Sie dann in Editoren oder im Eigenschaftenfenster bearbeiten.



StartUp	Der AutoLog-Modus kann über den manuellen Weg (durch einen Befehl in der SPS oder aus dem Konfigurator) oder automatisch beim Systemstart eingeschaltet werden.
Direction	Das eingestellte ADS-Gerät dient als Datenziel oder als Datenquelle.
Write Mode	Die Daten können in einer Datenbank zeilenweise angehängt, auf zeitlicher Basis oder nach Anzahl in einem Ringpuffer gehalten oder einfach an der entsprechenden Position aktualisiert werden.
Ringbuffer Parameter	Je nach Einstellung stellt dieser Parameter die Zeit oder die Zyklen dar, nach der der Ringbuffer aktualisiert wird.
Log Mode	Entweder wird die Variable nach Ablauf eines gewissen Zyklus oder auf Änderung geschrieben.
Cycle Time	Zykluszeit, nach welcher die Variable geschrieben wird.

Ads Device konfigurieren

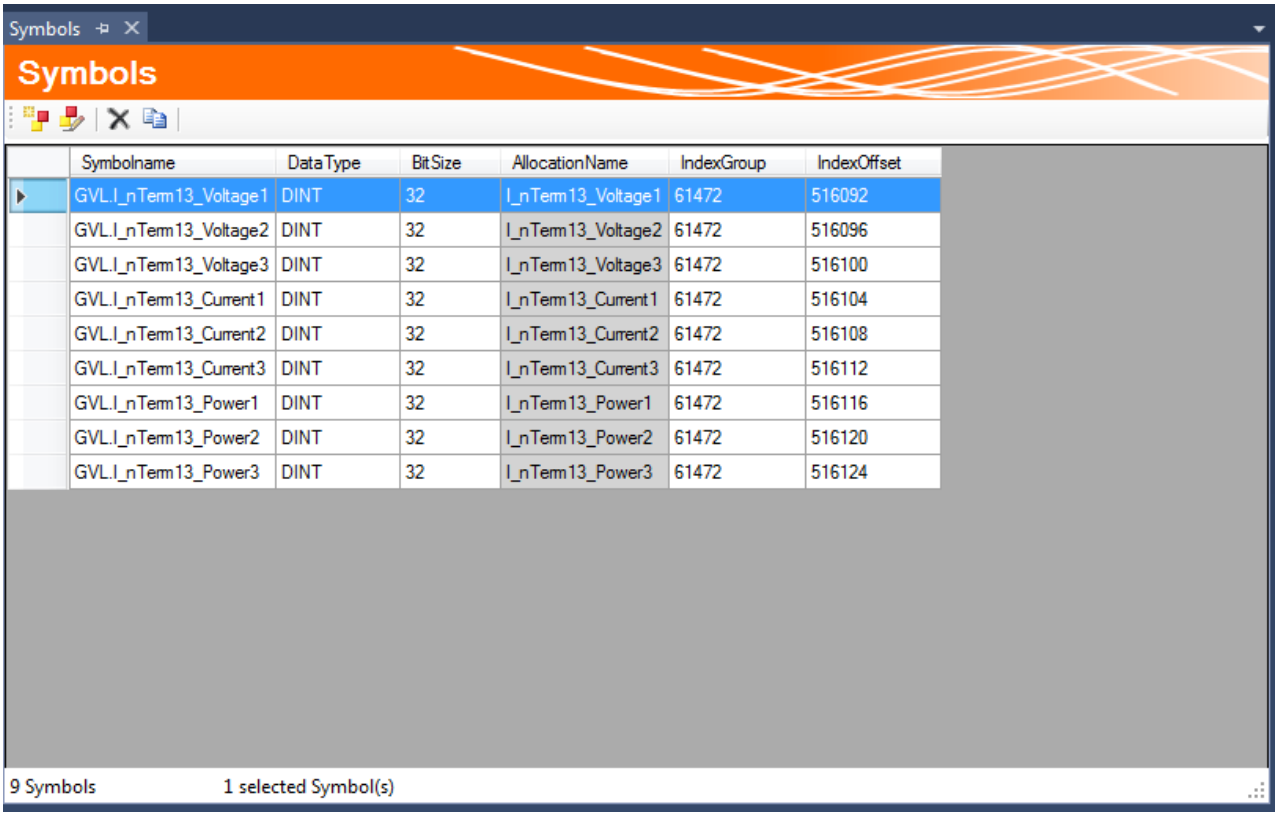
Das Ads Device wird automatisch unter eine AutoLog-Gruppe angelegt. Das Ads Device ist im häufigsten genutzten Anwendungsfall die SPS-Laufzeit. Folgende Einstellungen können im Editor getroffen werden:



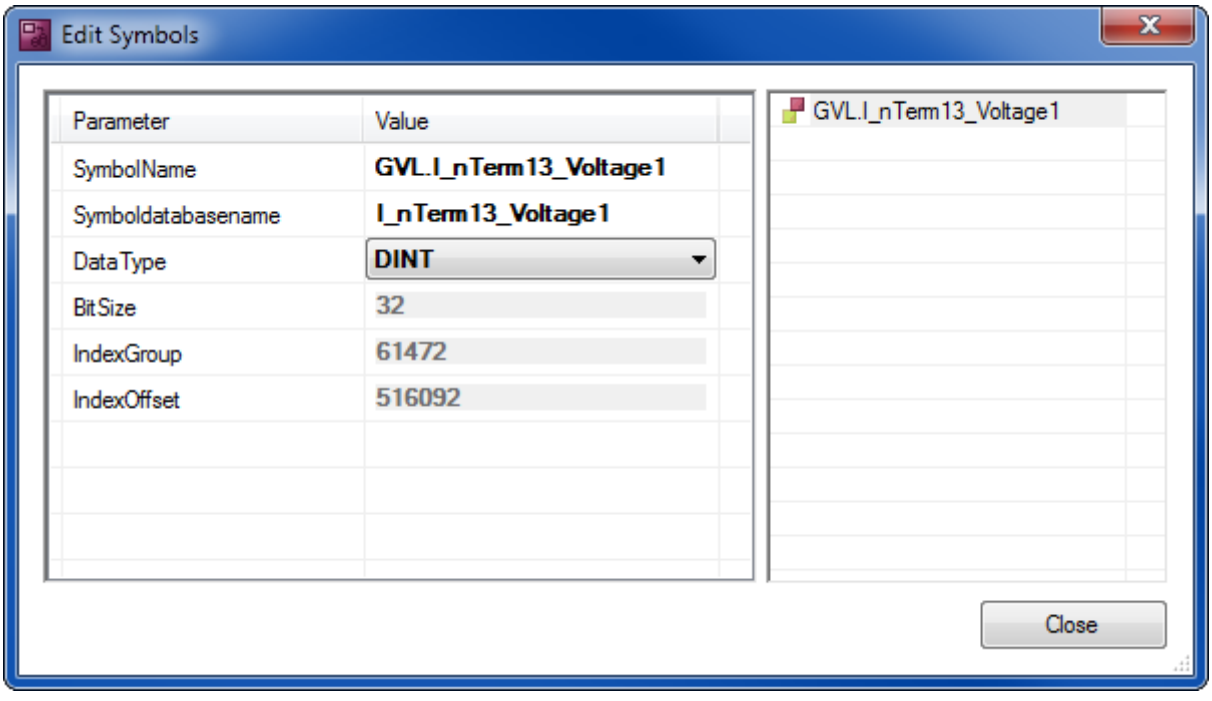
ADS Device	Bezeichnung des ADS-Zielgeräts.
AMS NetID	Adresse des Zielgeräts im TwinCAT-Netzwerk.
AMS Port	Port des Zielgeräts im TwinCAT-Netzwerk.
Timeout	Zeit, nach der von einem Verbindungsabbruch zum Zielgerät ausgegangen wird.
Connection Type	bySymbolName: Verbindung wird anhand des Namens des Symbols hergestellt. byIndexGroup: Verbindung wird anhand des Speicherindex hergestellt.

Symbole konfigurieren

Je nachdem, ob das ADS-Gerät Datenziel oder Datenquelle ist, werden die Symbole, die Sie hier einstellen, in die Datenbank geschrieben oder aus der Datenbank ausgelesen. Für einen komfortablen Zugriff können Sie den TwinCAT Target Browser [▶ 52] verwenden. Hier können die Symbole auf dem Target gesucht und per drag-and-drop zwischen den beiden Tools kommuniziert werden.



Sie könne Symbole auch manuell zur Symbolgruppe hinzufügen oder bearbeiten. Je nachdem, ob im ADS-Gerät der Verbindungstyp über die Symbolnamen oder die Index-Gruppen ausgewählt wurden, werden entsprechende Informationen benötigt. Dabei wird immer vom ADS-Gerät ausgegangen.



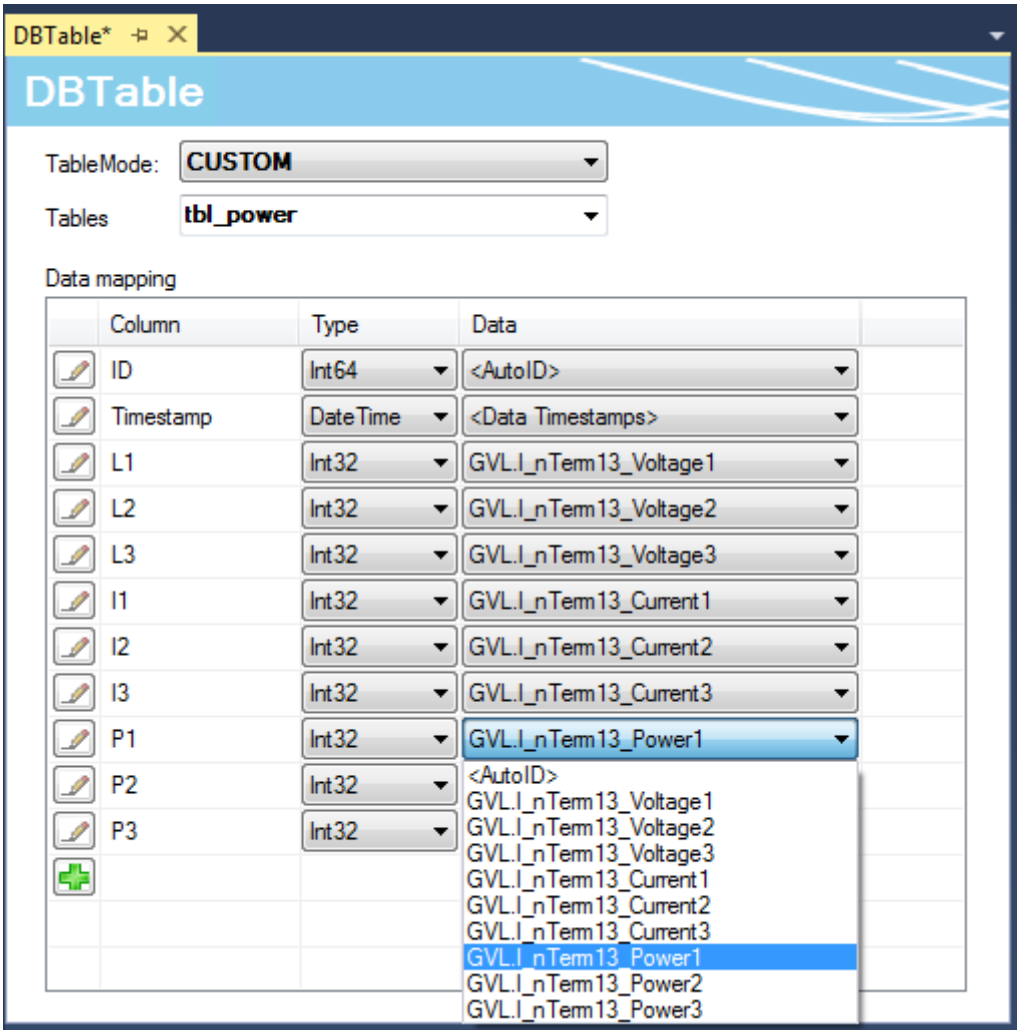
SymbolName	ausgehend vom einstellten ADS-Gerät wird das Symbol angesprochen
Symboldatabasename	Name der Variable in der Datenbanktabelle
DataType	SPS-Datentyp des Symbols
BitSize	Bit-Größe des Symbols (wird bei automatisch für die Datentypen eingestellt)
IndexGroup	Indexgruppe im TwinCAT-System
IndexOffset	Indexoffset im TwinCAT-System

Tabelle konfigurieren

Die Tabelle in einer Datenbank kann nach einer Standardtabellenstruktur oder nach einer individuellen Struktur aufgebaut sein.

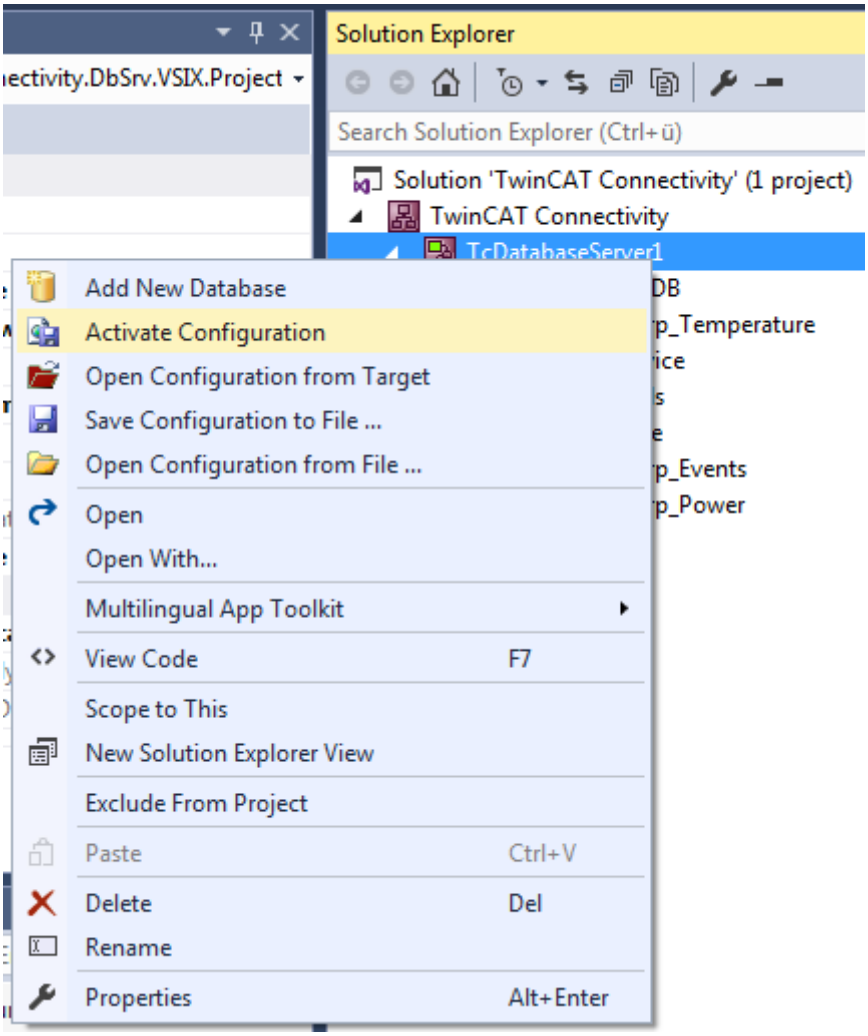
Die entsprechende Tabelle können Sie aus einer Liste möglicher Tabellen auswählen. Ist die Tabelle noch nicht vorhanden, können Sie diese mithilfe des SQL Query Editors erzeugen. Falls Sie die Standardtabellenstruktur auswählen, zeigt Ihnen ein blauer Haken an, ob die ausgewählte Tabelle dieser Struktur entspricht.

Der spezifische Tabellentyp bietet Ihnen die Möglichkeit, die einzelnen Symbole, welche in der Symbolgruppe eingestellt wurden, auf die Tabellenspalten in der Datenbank beliebig zu verteilen. Wird ein Datensatz nun während des AutoLog-Modus in die Datenbank geschrieben, werden die aktuellen Werte der Symbolgruppe zum Abtastzeitpunkt in der entsprechenden Spalte der Tabelle gespeichert.



Projekt aktivieren

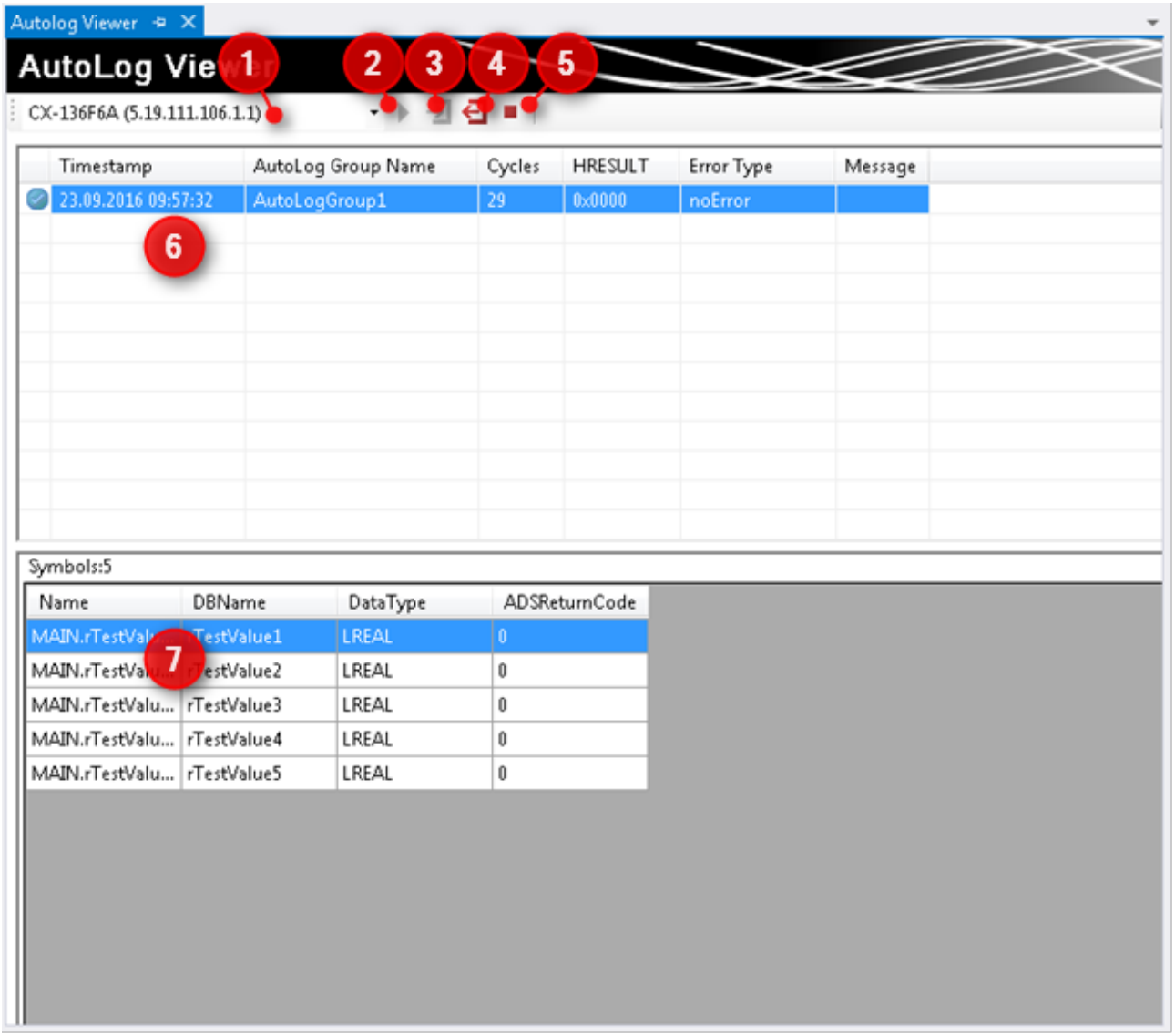
Um ein konfiguriertes Projekt auf dem TwinCAT Database Server zu aktivieren, verwenden Sie im Kontextmenü des TwinCAT-Database-Server-Projektes das Kommando **Activate Configuration**.



Je nachdem, welches Aufstartverhalten in der AutoLog-Gruppe angewiesen wurde, startet nun das Loggen der Variable mit dem Aufstarten des TwinCAT System. Manuell können Sie den Modus mithilfe des nachfolgenden AutologViewers oder mit dem entsprechenden Funktionsbaustein aus der SPS starten.

AutoLog Viewer

Der AutoLog Viewer des TwinCAT Database Server ist ein Tool, um den AutoLog-Modus zu steuern und zu überwachen. Ähnlich wie bei der TwinCAT SPS können Sie sich auf ein Zielsystem einloggen. Im eingeloggten Zustand kann der AutoLog-Modus gestartet oder gestoppt werden. Im unteren Bereich des Fensters werden Informationen über den aktuellen Zustands des Loggens mitgeteilt. Durch das Selektieren einer AutoLog-Gruppe werden weitere Informationen über die geloggten Symbole angezeigt.



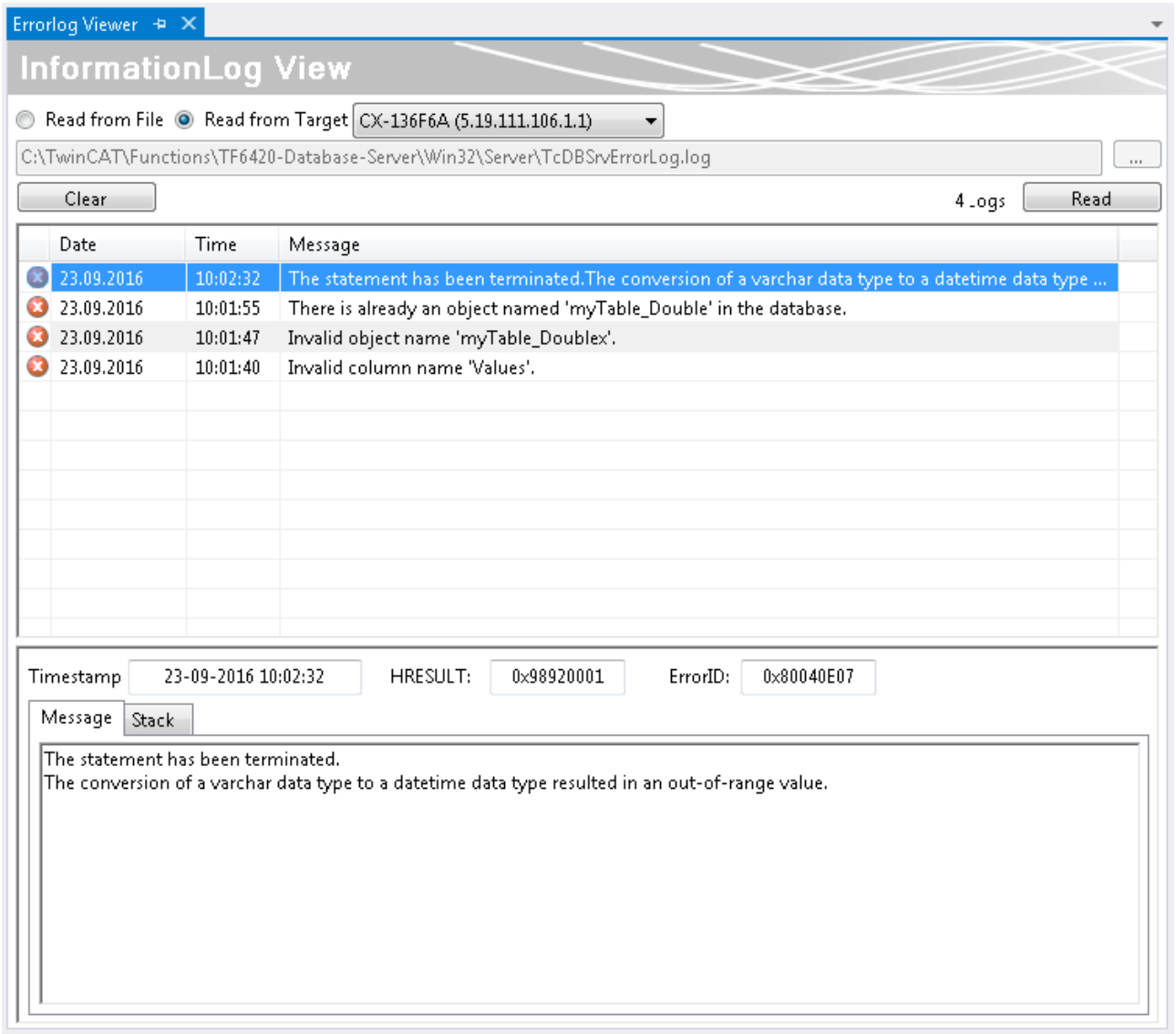
ID	Bezeichnung	Funktion
1	Zielsystem	Auswahl des Zielsystems mit installiertem TwinCAT Database Server
2	Start	Manueller Start des AutoLog-Modus
3	Einloggen	Einloggen auf den aktiven AutoLog-Prozess
4	Ausloggen	Ausloggen aus dem aktiven AutoLog-Prozess
5	Stopp	Manueller Stopp des AutoLog-Modus
6	Autologgruppen	Auflistung konfigurierter Autolog-Gruppen auf dem Zielsystem
7	Symbole	Auflistung konfigurierter Symbole der ausgewählten AutoLog-Gruppe

Mit dem Autolog Viewer kann die konfigurierte Applikation gestartet und überwacht werden. Je nach Einstellung ist nach dem Einloggen und Starten der hochzählende Zykluszähler der AutoLog-Gruppen entsprechend der Aktualisierungszeiten sichtbar. Auch Fehler beim Aktualisieren können Sie hier feststellen. Zur detaillierteren Behandlung empfehlen wir den InformationLog View.

Nähere Fehlerbehandlung mit dem InformationLog View

Der InformationLog View ist ein Tool, um die Logdateien vom TwinCAT Database Server auszulesen. Protokollierte Informationen werden mit einem Zeitstempel, IDs und Fehlermeldungen im Klartext angezeigt.

Die Log-Dateien können nicht nur über den direkten Dateizugriff eingesehen oder geleert werden, sondern auch direkt über das Target. Gerade für verteilte Database Server im Netzwerk ist dies vorteilhaft, um einen schnellen und einfachen Zugriff auf die Logdatei zu erlangen. Für diesen Zugriff muss eine Route zum Zielgerät bestehen.



5.1.1.3 PLC Expert Mode

Dieses Kapitel ist eine Zusammenstellung aller nötigen Informationen, um den SPS-Expertenmodus (PLC Expert Mode) des TwinCAT Database Servers zu nutzen. Anders als im Configure Mode werden Daten in diesem Modus nicht zyklisch-/eventbezogen, sondern zu spezifischen Zeitpunkten im Programmablauf geschrieben oder gelesen. Dafür werden keine Kenntnisse aus der SQL-Sprache benötigt.

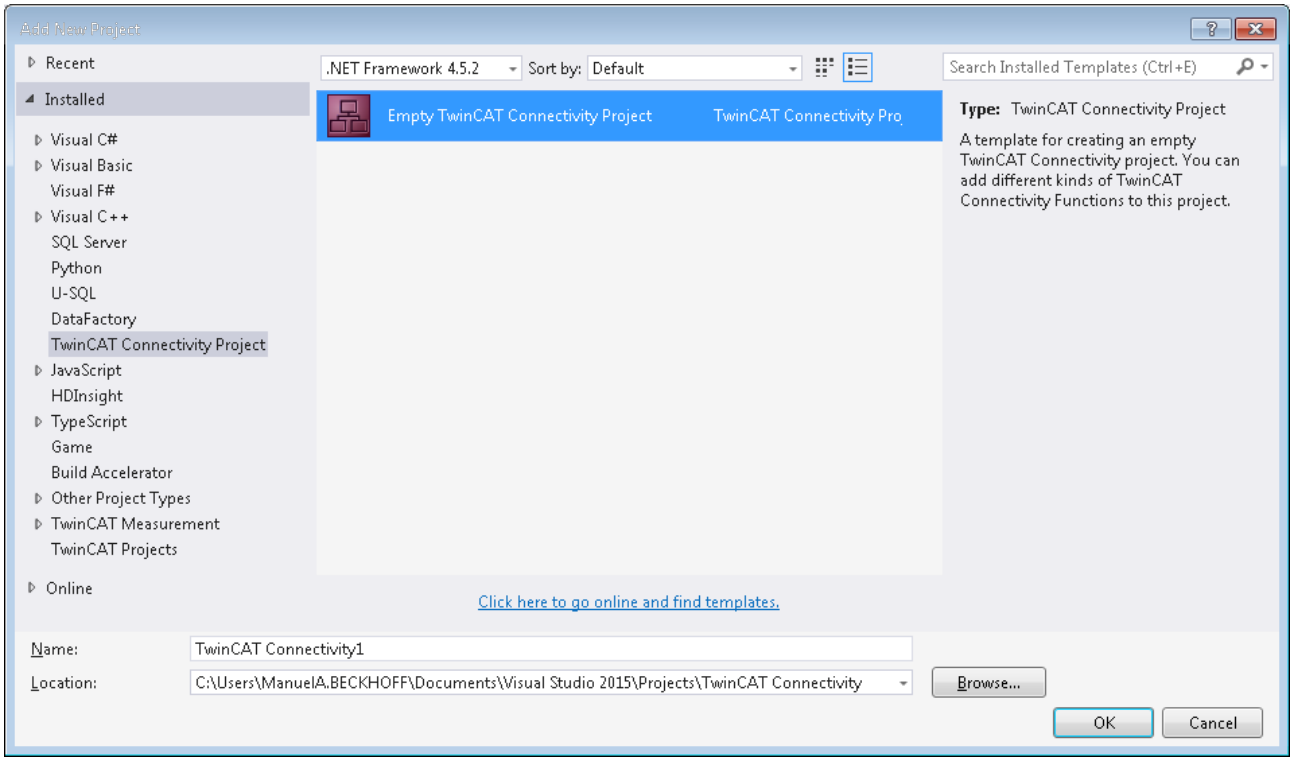
PLC Expert Mode

Im PLC Expert Mode wird nur die Datenbankkonfiguration im Konfigurator eingestellt. Weitere Funktionalitäten werden im SPS-Code der Applikation implementiert. Mit dem Funktionsbaustein FB_PLCDBCreate [▶ 175] ist es sogar möglich, ohne den Konfigurator auszukommen und selbst die Datenbank aus der SPS heraus zu konfigurieren. Ansonsten stehen Funktionsbausteine zum Lesen und Schreiben zur Verfügung. Der Funktionsbaustein FB_PLCDBCmd [▶ 187] bildet dabei den Übergang zwischen PLC Expert Mode und SQL Expert Mode. Hier können Tabellenstrukturen sehr einfach als SPS-Struktur abgebildet und ein SQL-Kommando mit Platzhaltern für die aktuellen Werte der Struktur an den TwinCAT Database Server übergeben werden. Der TwinCAT Database Server setzt dann selbständig alle Werte ein und schickt das Kommando an die Datenbank.

Projekt erstellen

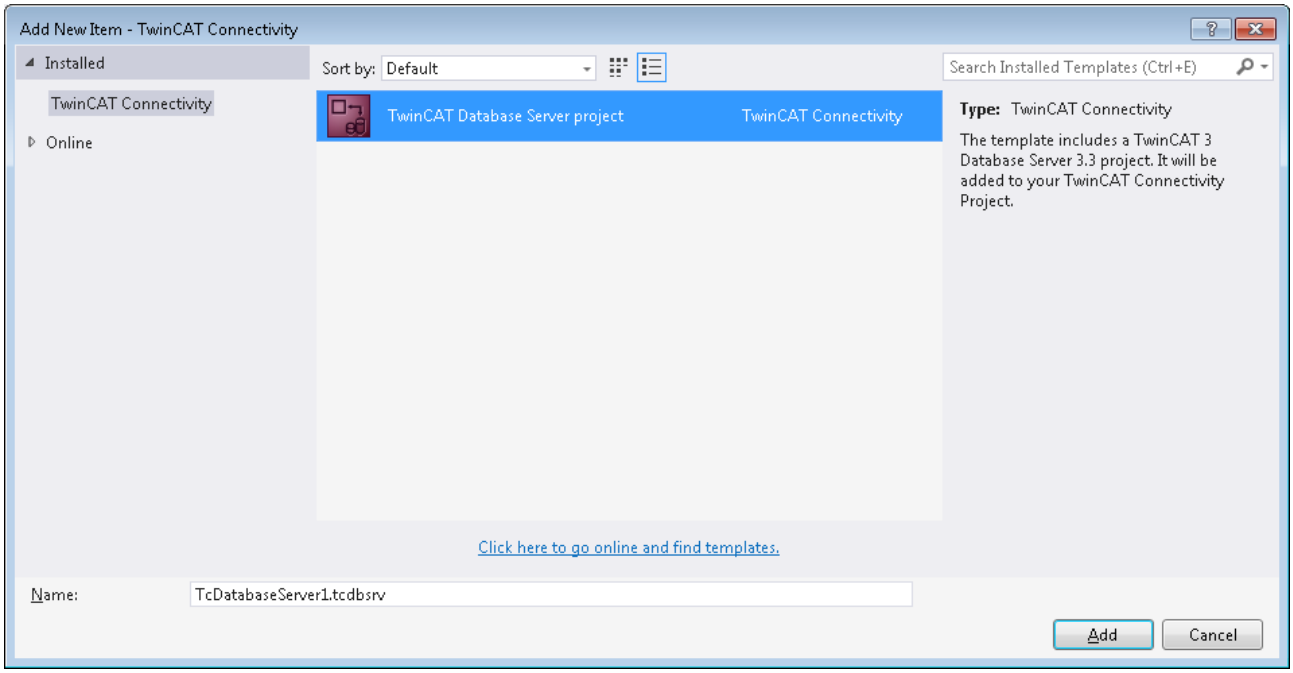
Durch die TwinCAT-Connectivity-Erweiterung für Visual Studio steht eine neue Projektvorlage zur Verfügung. Beim Erstellen eines neuen Projektes erscheint nun die Kategorie **TwinCAT Connectivity Project** in der Auswahl.

Um ein neues TwinCAT-Connectivity-Projekt zu erstellen, wählen Sie das **Empty TwinCAT Connectivity Project**, legen den Projektnamen und den Speicherort fest und fügen es mit **OK** der Solution hinzu. TwinCAT-Connectivity-Projekte bzw. TwinCAT-Database-Server-Projekte können so komfortabel neben TwinCAT- oder anderen Visual-Studio-Projekten angelegt werden.

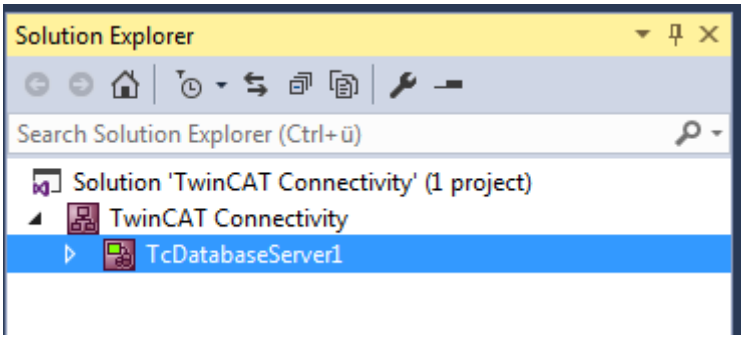


In der Solution erscheint ein neuer Projektknoten. Unterhalb des Connectivity-Projektknotens können Sie Subprojekte der unterstützten Connectivity-Funktionen ergänzen.

Mit **Add** können Sie dem TwinCAT-Connectivity-Projekt ein neues TwinCAT-Database-Server-Projekt hinzufügen. Das TwinCAT-Database-Server-Projekt befindet sich in der Auflistung der vorhandenen Item Templates.



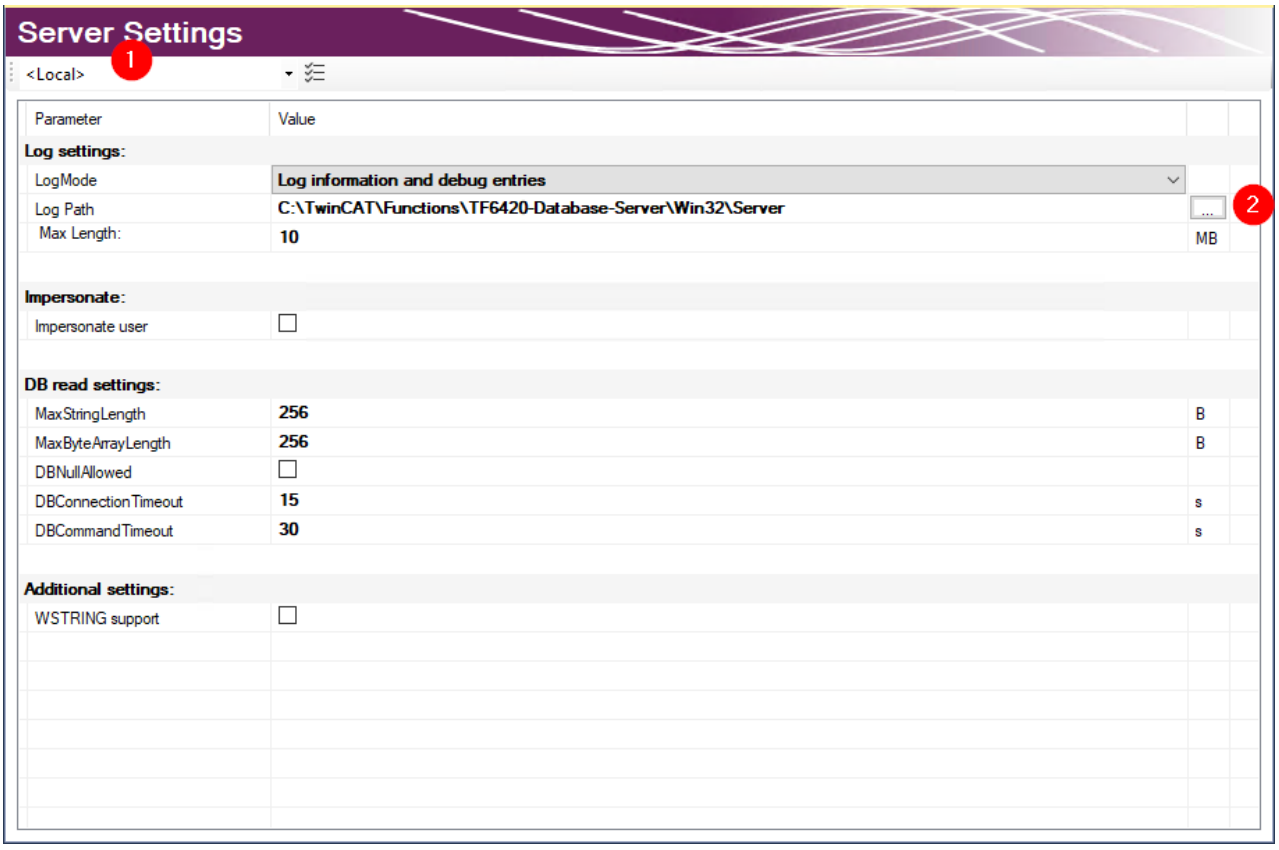
Unterhalb des TwinCAT-Connectivity-Knotens wird ein neues TwinCAT-Database-Server-Projekt angelegt.



Dieses dient nun als Basis für die anstehende Konfiguration eines TwinCAT Database Servers. Das Dokument können Sie sowohl über die Eigenschaften im Eigenschaftsfenster, als auch über einen Editor bearbeiten.

Einem Connectivity-Projekt können Sie beliebig viele TwinCAT-Database-Server-Projekte oder andere Projekte hinzufügen, und damit auch mehrere Konfigurationen in einem Connectivity-Projekt einstellen.

Editor für Server-Einstellungen



Im Editor **Server Settings** können Sie die Einstellungen für den TwinCAT Database Server bearbeiten. Diese Einstellungen betreffen den entsprechenden Server im Allgemeinen. Im Drop-down-Menü (1) können Sie das Zielsystem über die Ams-NetId angegeben. Dafür müssen Sie über TwinCAT eine Route zum Zielsystem anlegen. Wird eine fertige Konfiguration übertragen, werden die Einstellungen auf den TwinCAT Database Server dieses Zielsystems gespeichert.

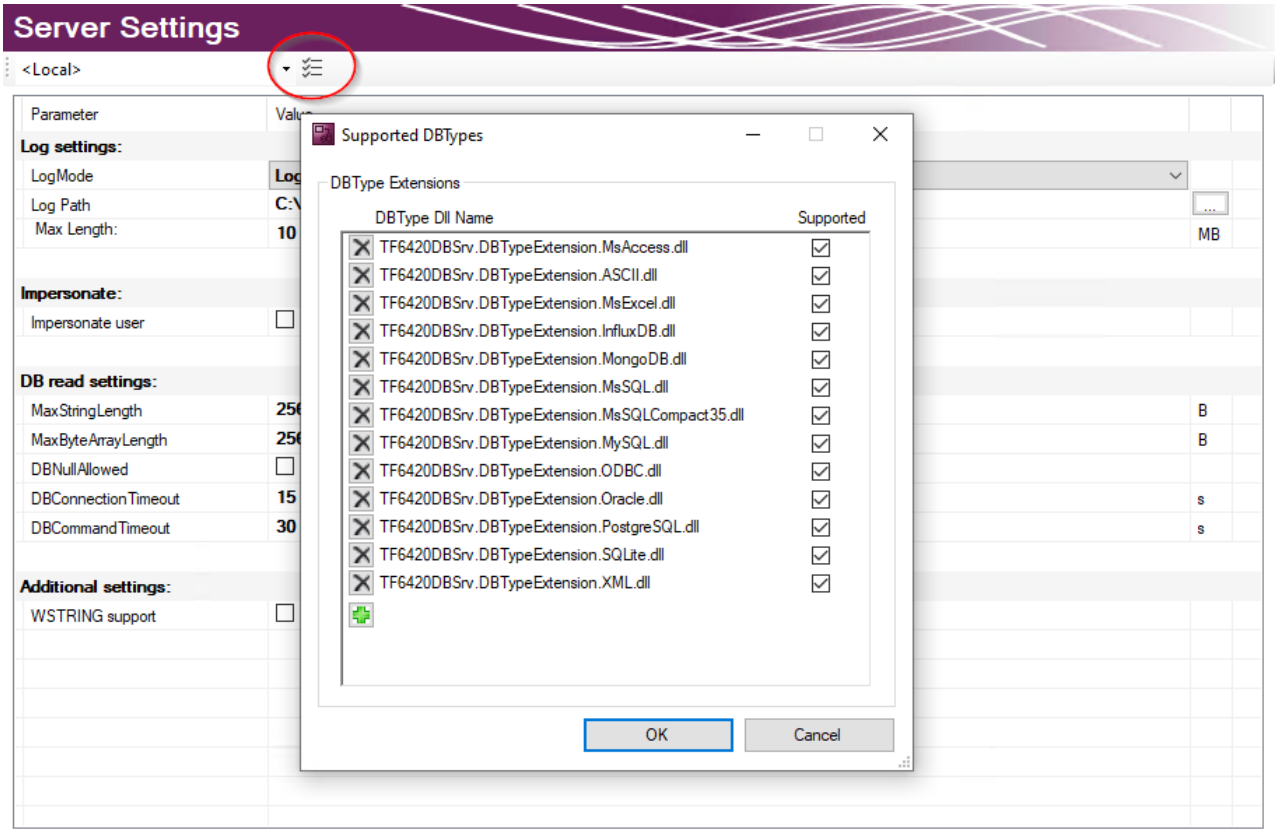
In den **Log settings** konfigurieren Sie Einstellungen zum Aufzeichnen von Fehlerfällen. In einem Fehlerfall erzeugt der Database Server einen detaillierten Eintrag in einer Logdatei. Diese können Sie mit dem Information Log Viewer [▶ 51] auslesen. In den **Log Settings** geben Sie einen Pfad zum Ablageort und die maximale Größe der Datei an. Zusätzlich können Sie die Genauigkeit des Logs beeinflussen. Wir empfehlen, aus Performancegründen, das Loggen nach erfolgter Fehleranalyse wieder abzuschalten, wenn es nicht mehr benötigt wird.

Bei Netzwerkzugriff auf dateibasierende Datenbanken wie Access Datenbanken oder SQL Compact Datenbanken müssen Sie die Option **Impersonate** setzen, damit sich der TwinCAT Database Server mit dem Netzwerklaufwerk verbinden kann. **Diese Funktion wird zurzeit nicht unter Windows CE unterstützt.**

Sie können weitere Einstellungen konfigurieren, um das Lesen aus der Datenbank zu steuern. Diese Einstellungen beziehen sich auf den TwinCAT Database Server auf dem Zielsystem:

MaxStringLength	Maximale String-Länge der Variablen in der SPS
MaxByteArrayLength	Maximale Byte-Array-Länge der Variablen in der SPS
DBNullAllowed	Gibt an, ob NULL-Werte im TwinCAT Database Server angenommen werden.
DBConnectionTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem Verbindungsaufbau von einem Verbindungsfehler ausgeht.
DBCommandTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem abgesendeten Kommando von einem Verbindungsfehler ausgeht. Bei hohen Datenmengen kann, je nach Datenbank und Infrastruktur ein Befehl durchaus sehr viel Zeit in Anspruch nehmen.

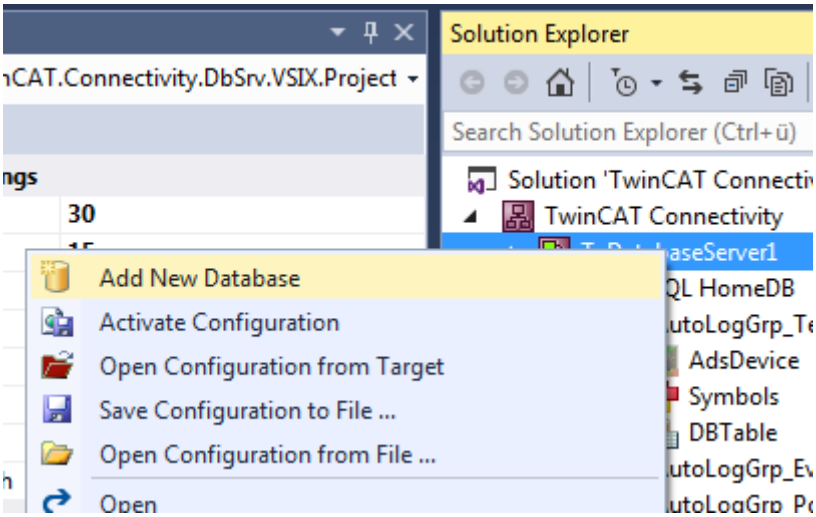
Unterstützte Datenbanktypen



Die installierten Datenbanktypen können in den Server Einstellungen ausgewählt werden. Standardmäßig sind alle installierten Datenbanken angewählt. Der TwinCAT 3 Database Server wird die entsprechenden Datenbankschnittstellen laden. Auf diese Weise können nicht genutzte Datenbanken auf dem Zielsystem abgewählt werden.

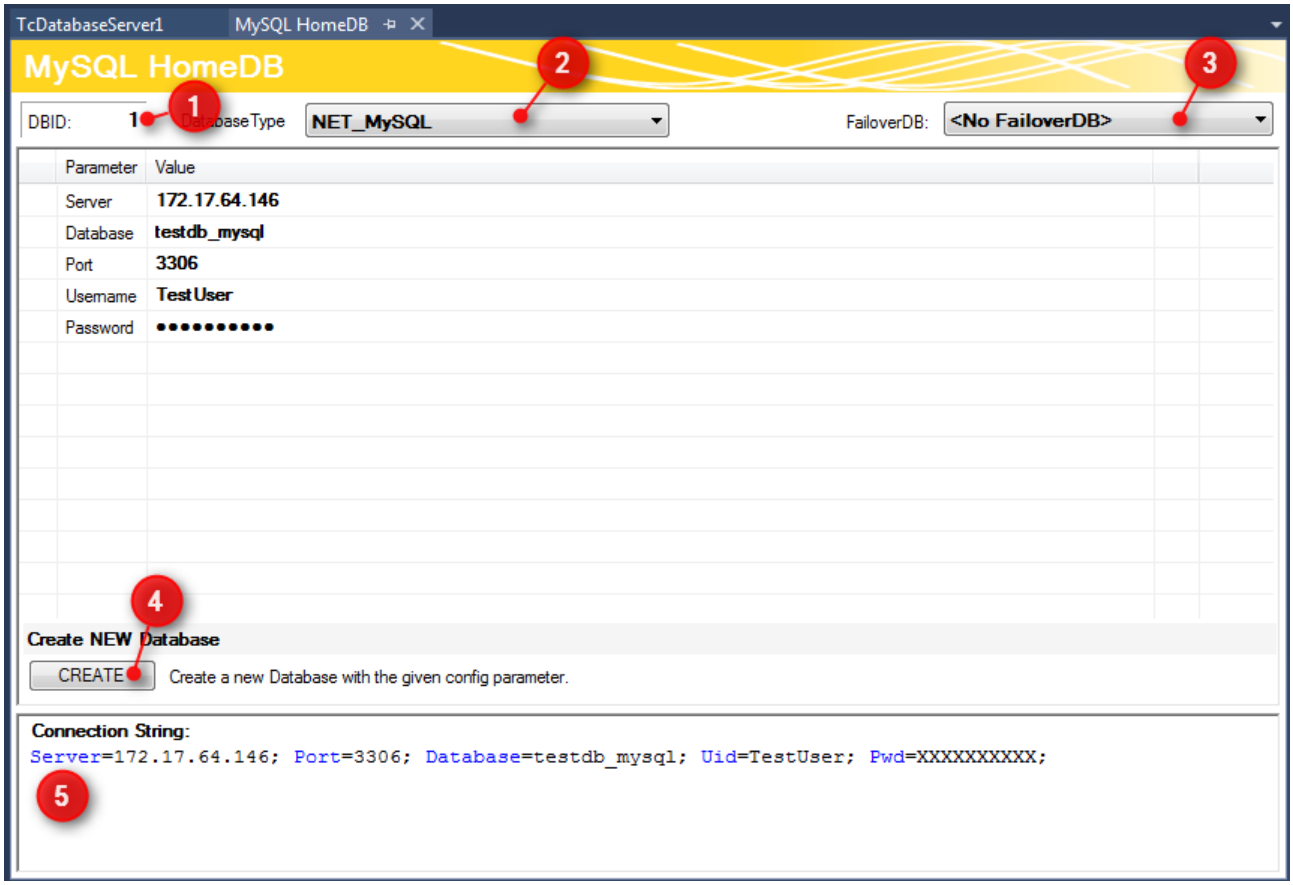
Datenbankkonfiguration hinzufügen

Eine neue Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new Database** über das Kontextmenü eines Database-Server-Projektes oder das entsprechende Kommando in der Toolbar hinzufügen.



Eine neue Datenbankkonfiguration wird als Datei auf im Projektordner hinzugefügt und in das Projekt eingebunden. Wie bei allen Visual-Studio-Projekten, werden die Informationen über die neuen Dateien im Connectivity-Projekt hinterlegt.

Editor für Datenbankkonfigurationen

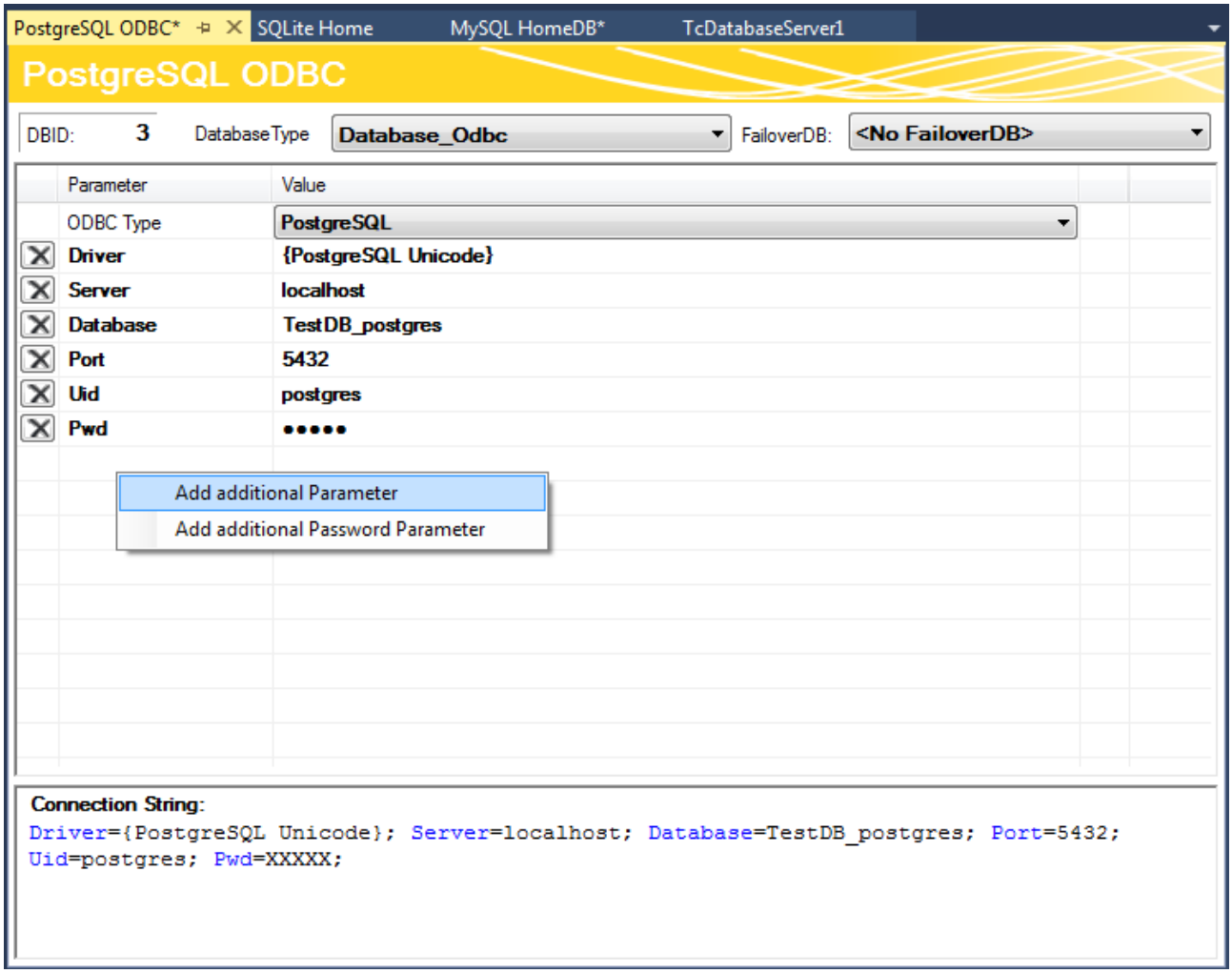


Im oberen Teil des Editors wird die Datenbank ID angezeigt (1), die für einige Funktionsbausteine in der SPS benötigt wird. Den Datenbanktyp der Zieldatenbank können Sie aus dem Drop-down-Menü auswählen (2). Hier können Sie auch die Odbc-Schnittstelle für eine Datenbank nutzen, die noch nicht unterstützt wird. Beachten Sie, dass je nach Datenbank nicht alle Funktionen des TwinCAT Database Servers gewährleistet werden.

Außerdem können Sie optional eine sogenannte FailOver-Datenbank (3) auswählen, welche im ‚Configure'-Mode beim Fehlerfall einspringt. Bei einem Verbindungsabbruch zum Netzwerk kann in einem solchen Fall automatisch gewährleistet werden, dass keine Daten verloren gehen und an anderer Stelle gespeichert werden.

Für jede Datenbank [▶ 126] stehen zusätzlich weitere einstellbare Parameter zur Verfügung. Je nach Datenbank wird ein Connection String (5) erstellt, welcher die Verbindung zur Datenbank beschreibt. Diese Anzeige dient zur Transparenz Ihrer eingestellten Parameter.

Über die Schaltfläche **CREATE** (4) können Sie eine neue Datenbank erstellen. Diese Funktion wird nur angezeigt, wenn sie von der jeweiligen Datenbank unterstützt wird.

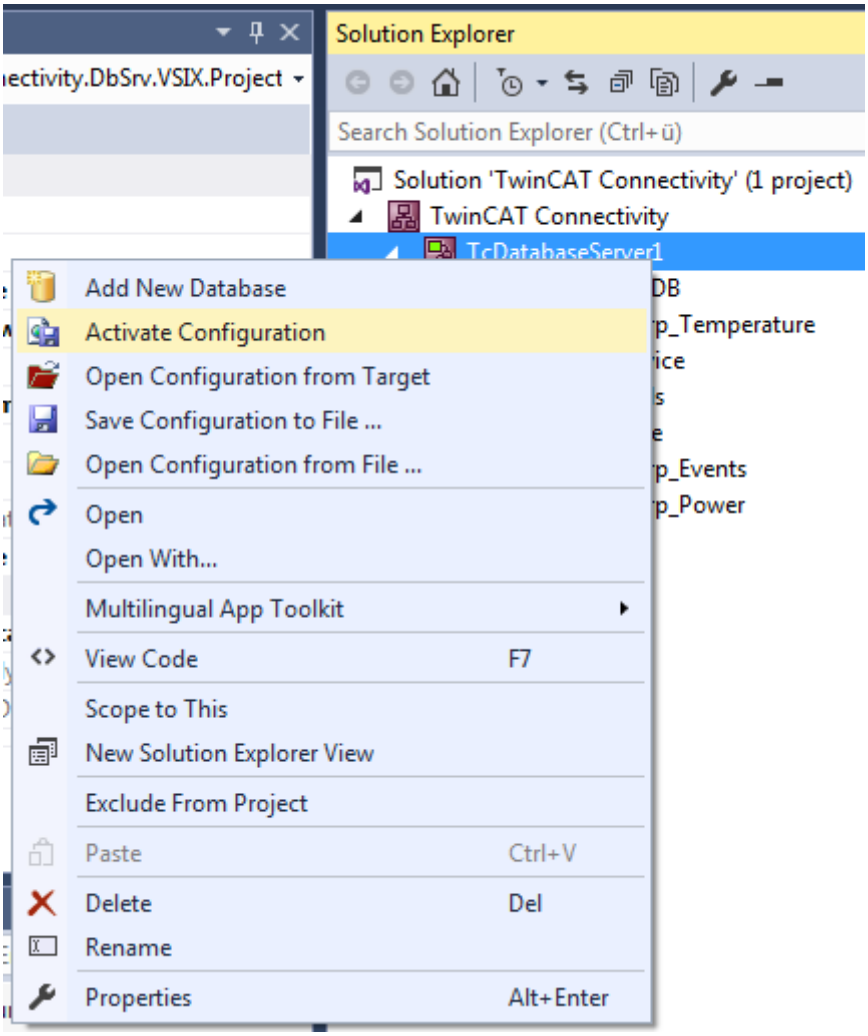


Sie können auch unbekannte Datenbanken mit einer ODBC-Schnittstelle konfigurieren. Dafür wählen Sie im in der Drop-down-Liste **ODBC Type** den Eintrag „Unknown Database“ und fügen über die Befehle im Kontextmenü Parameter hinzu. Diese können auch Passwörter beinhalten, welche dann verschlüsselt abgespeichert werden. Daraus kann der gewünschte Connection String zusammengestellt werden. Beachten Sie, dass nur begrenzte Funktionen des TwinCAT Database Servers genutzt werden können. Nur die expliziten Funktionsbausteine des SQL Expert Modes werden unterstützt.

Eine zusätzliche AutoLog-Gruppen-Konfiguration ist in diesem Modus nicht notwendig, da das Schreiben und Lesen zwischen Datenbank und der SPS manuell vom SPS-Programmierer aufgerufen wird. Der Konfigurationsteil ist damit abgeschlossen.

Projekt aktivieren

Um ein konfiguriertes Projekt auf dem TwinCAT Database Server zu aktivieren, verwenden Sie im Kontextmenü des TwinCAT-Database-Server-Projektes das Kommando **Activate Configuration**.



Mit dem SQL Query Editor new können Sie nach Aktivierung des Projektes weitere Entwicklungsschritte tätigen, wie zum Beispiel Datenbanken oder Tabellen angelegen, Strukturen für die SPS generieren, welche der passenden Tabellenstruktur der Datenbank entsprechen, oder Verbindungen zur Datenbank mit den getätigten Informationen testen.

Um den TwinCAT Database Server nun anzusprechen kann der SPS-Programmierer die zur Verfügung stehenden Funktionsbausteine der SPS API [▶ 168] benutzen.

5.1.1.4 SQL Expert Mode

Dieses Kapitel umfasst alle nötigen Schritte, um den SQL Expertenmodus (SQL Expert Mode) zu benutzen. Dieser Modus ist für den Benutzer mit individuellen Anforderungen zugeschnitten. Folgende Themen werden behandelt:

1. Erstellen eines Projektes
2. Anlegen und Einstellen einer Datenbankkonfiguration
3. Aktivieren eines Database Server Projektes
4. Erstellen von SQL Befehlen mithilfe des SQL Query Editors

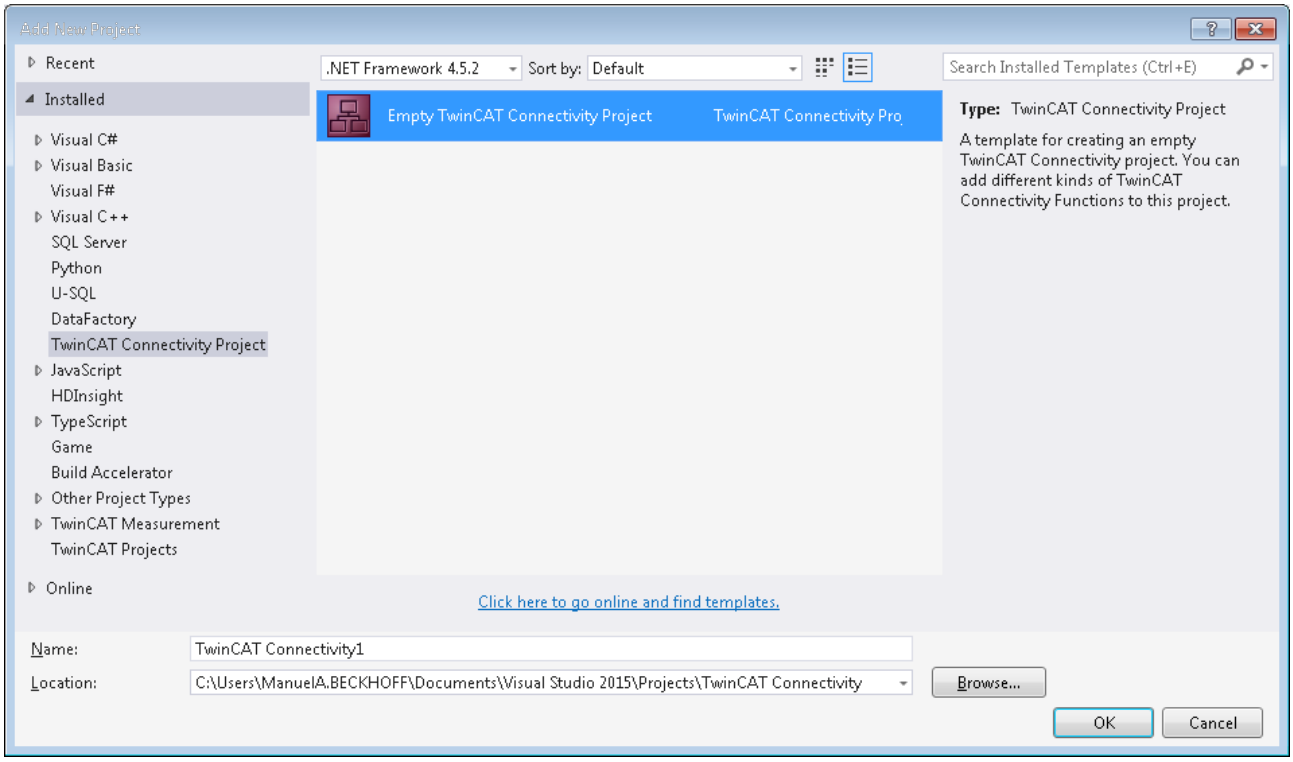
SQL Expert Mode

Im SQL Expert Mode kann sich der Anwender die SQL-Kommandos für zum Beispiel Insert, Select oder Update selber in der SPS zusammenbauen und über den TwinCAT Database Server zur Datenbank schicken. Ein sehr flexibler und performanter Weg. Auch sogenannte Stored Procedures [▶ 205], welche in der Datenbank hinterlegt sind, können aus der SPS aufgerufen werden.

Projekt erstellen

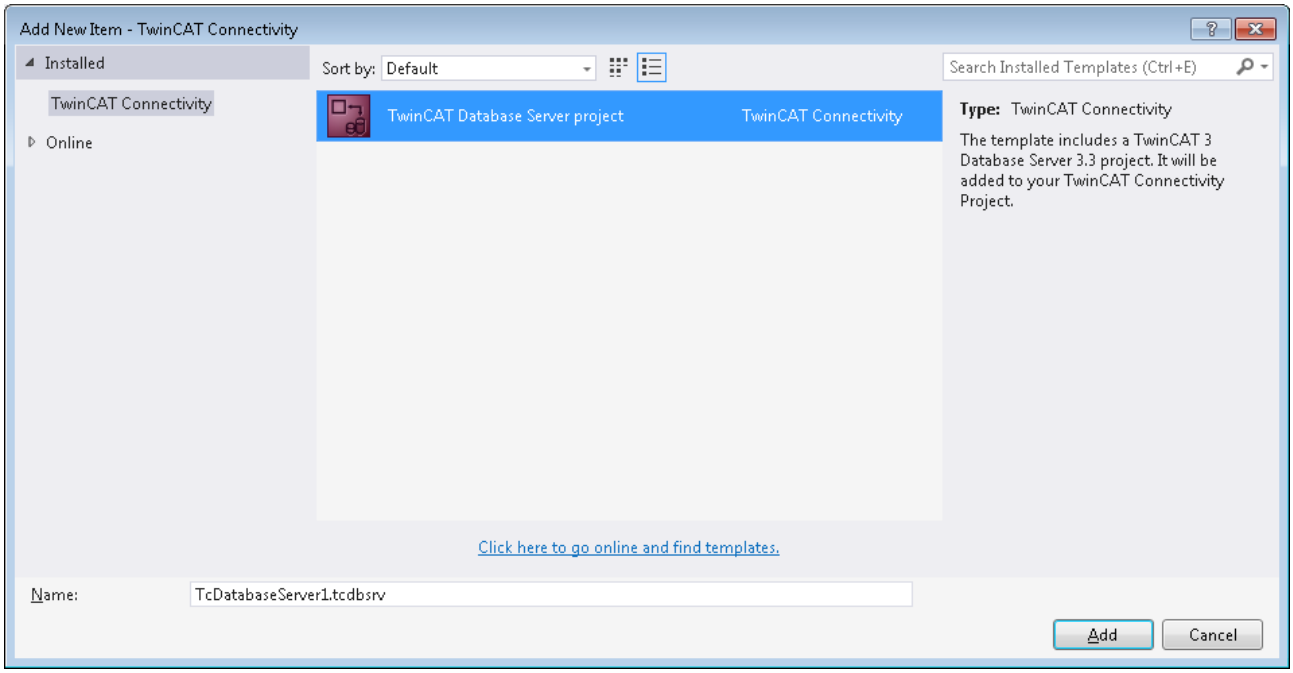
Durch die TwinCAT-Connectivity-Erweiterung für Visual Studio steht eine neue Projektvorlage zur Verfügung. Beim Erstellen eines neuen Projektes erscheint nun die Kategorie **TwinCAT Connectivity Project** in der Auswahl.

Um ein neues TwinCAT-Connectivity-Projekt zu erstellen, wählen Sie das **Empty TwinCAT Connectivity Project**, legen den Projektnamen und den Speicherort fest und fügen es mit **OK** der Solution hinzu. TwinCAT-Connectivity-Projekte bzw. TwinCAT-Database-Server-Projekte können so komfortabel neben TwinCAT- oder anderen Visual-Studio-Projekten angelegt werden.

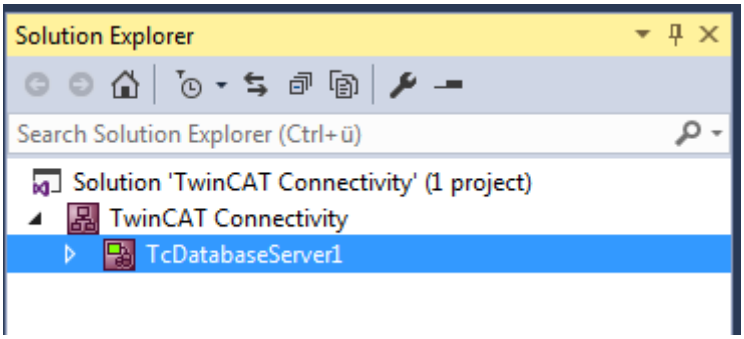


In der Solution erscheint ein neuer Projektknoten. Unterhalb des Connectivity-Projektknotens können Sie Subprojekte der unterstützten Connectivity-Funktionen ergänzen.

Mit **Add** können Sie dem TwinCAT-Connectivity-Projekt ein neues TwinCAT-Database-Server-Projekt hinzufügen. Das TwinCAT-Database-Server-Projekt befindet sich in der Auflistung der vorhandenen Item Templates.



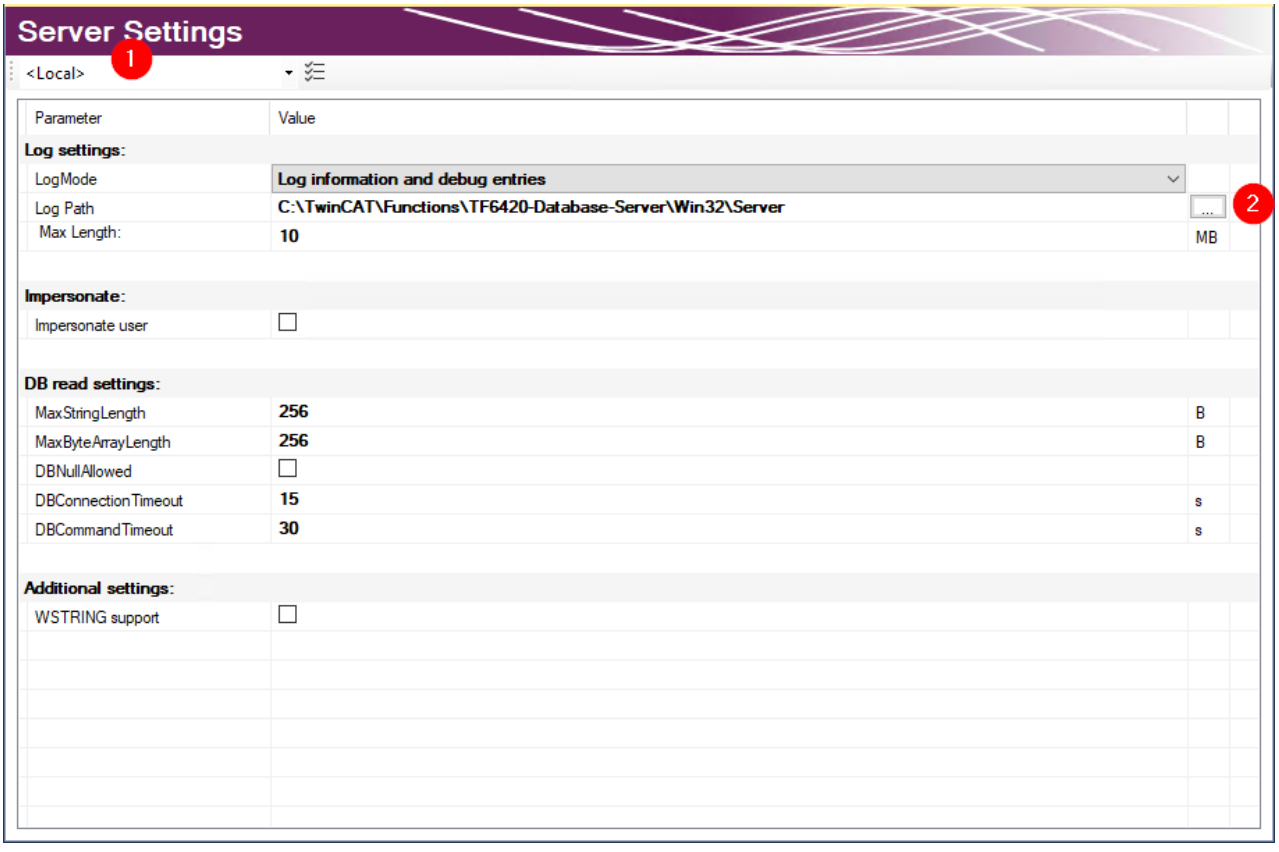
Unterhalb des TwinCAT-Connectivity-Knotens wird ein neues TwinCAT-Database-Server-Projekt angelegt.



Dieses dient nun als Basis für die anstehende Konfiguration eines TwinCAT Database Servers. Das Dokument können Sie sowohl über die Eigenschaften im Eigenschaftsfenster, als auch über einen Editor bearbeiten.

Einem Connectivity-Projekt können Sie beliebig viele TwinCAT-Database-Server-Projekte oder andere Projekte hinzufügen, und damit auch mehrere Konfigurationen in einem Connectivity-Projekt einstellen.

Editor für Server-Einstellungen



Im Editor **Server Settings** können Sie die Einstellungen für den TwinCAT Database Server bearbeiten. Diese Einstellungen betreffen den entsprechenden Server im Allgemeinen. Im Drop-down-Menü (1) können Sie das Zielsystem über die Ams-NetId angegeben. Dafür müssen Sie über TwinCAT eine Route zum Zielsystem anlegen. Wird eine fertige Konfiguration übertragen, werden die Einstellungen auf den TwinCAT Database Server dieses Zielsystems gespeichert.

In den **Log settings** konfigurieren Sie Einstellungen zum Aufzeichnen von Fehlerfällen. In einem Fehlerfall erzeugt der Database Server einen detaillierten Eintrag in einer Logdatei. Diese können Sie mit dem Information Log Viewer [▶ 51] auslesen. In den **Log Settings** geben Sie einen Pfad zum Ablageort und die

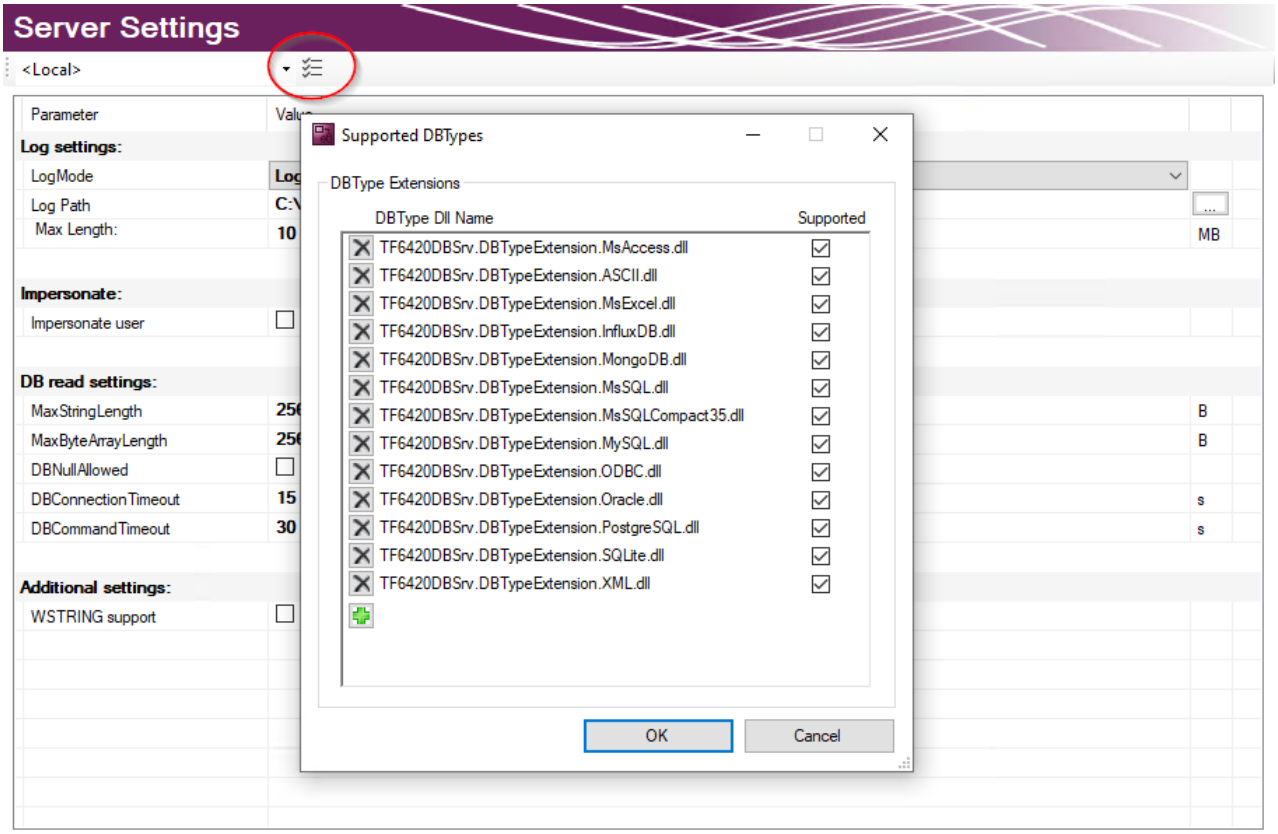
maximale Größe der Datei an. Zusätzlich können Sie die Genauigkeit des Logs beeinflussen. Wir empfehlen, aus Performancegründen, das Loggen nach erfolgter Fehleranalyse wieder abzuschalten, wenn es nicht mehr benötigt wird.

Bei Netzwerkzugriff auf dateibasierende Datenbanken wie Access Datenbanken oder SQL Compact Datenbanken müssen Sie die Option **Impersonate** setzen, damit sich der TwinCAT Database Server mit dem Netzwerklaufwerk verbinden kann. **Diese Funktion wird zurzeit nicht unter Windows CE unterstützt.**

Sie können weitere Einstellungen konfigurieren, um das Lesen aus der Datenbank zu steuern. Diese Einstellungen beziehen sich auf den TwinCAT Database Server auf dem Zielsystem:

MaxStringLength	Maximale String-Länge der Variablen in der SPS
MaxByteArrayLength	Maximale Byte-Array-Länge der Variablen in der SPS
DBNullAllowed	Gibt an, ob NULL-Werte im TwinCAT Database Server angenommen werden.
DBConnectionTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem Verbindungsaufbau von einem Verbindungsfehler ausgeht.
DBCommandTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem abgesendeten Kommando von einem Verbindungsfehler ausgeht. Bei hohen Datenmengen kann, je nach Datenbank und Infrastruktur ein Befehl durchaus sehr viel Zeit in Anspruch nehmen.

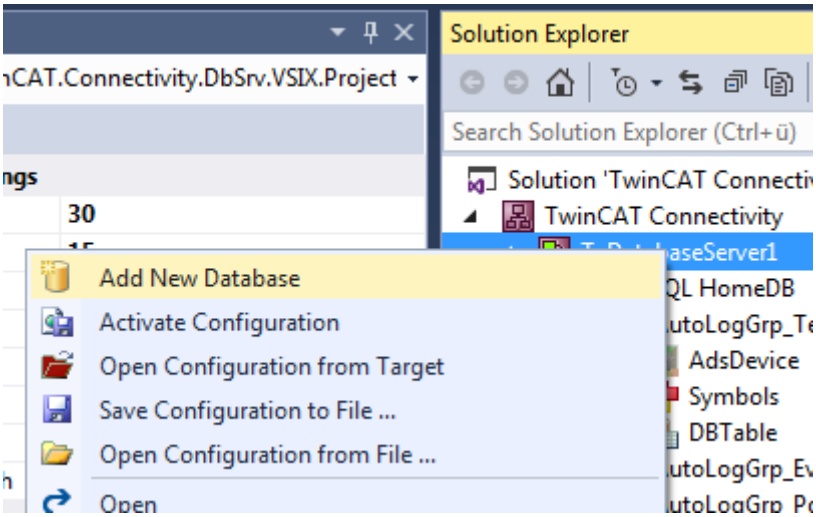
Unterstützte Datenbanktypen



Die installierten Datenbanktypen können in den Server Einstellungen ausgewählt werden. Standardmäßig sind alle installierten Datenbanken angewählt. Der TwinCAT 3 Database Server wird die entsprechenden Datenbankschnittstellen laden. Auf diese Weise können nicht genutzte Datenbanken auf dem Zielsystem abgewählt werden.

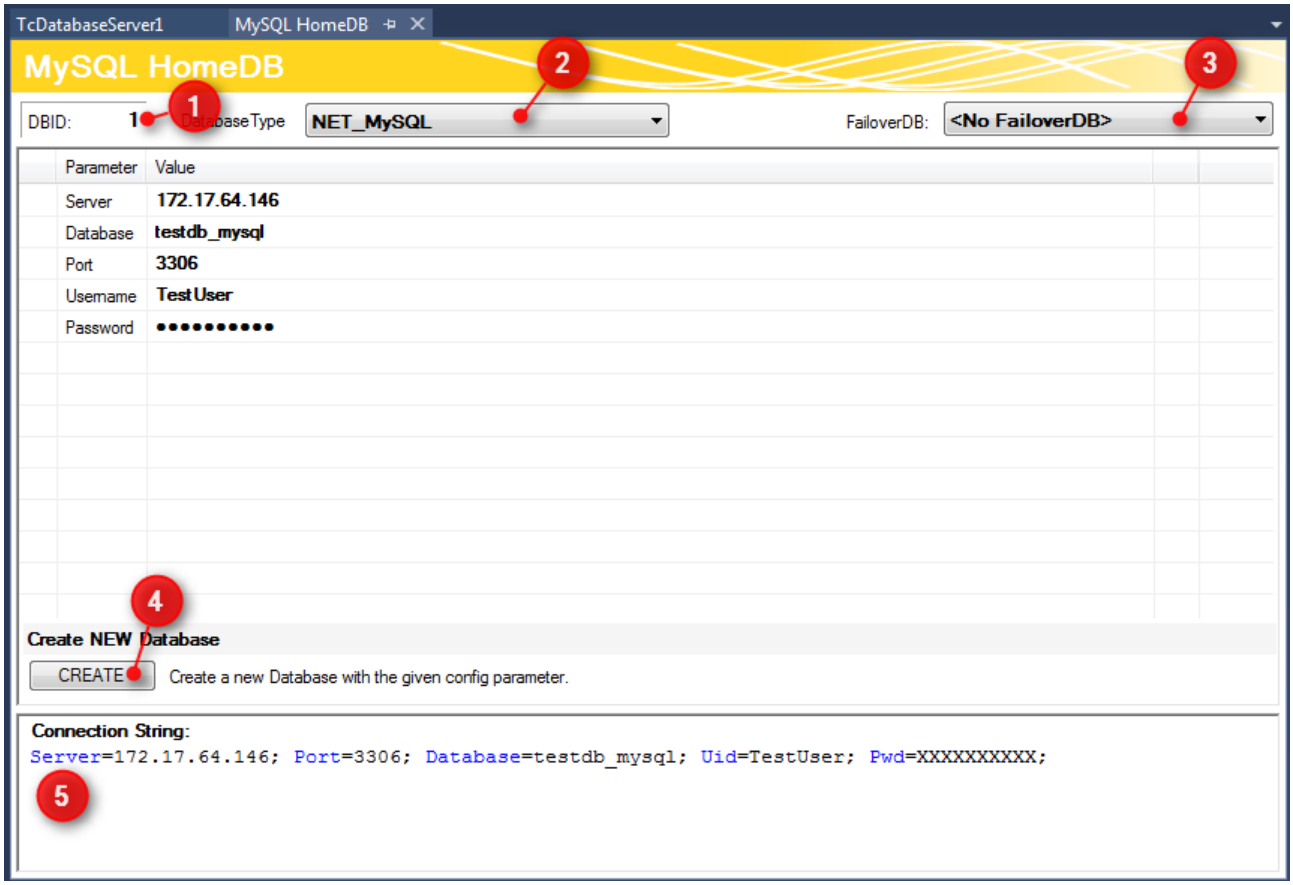
Datenbankkonfiguration hinzufügen

Eine neue Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new Database** über das Kontextmenü eines Database-Server-Projektes oder das entsprechende Kommando in der Toolbar hinzufügen.



Eine neue Datenbankkonfiguration wird als Datei auf im Projektordner hinzugefügt und in das Projekt eingebunden. Wie bei allen Visual-Studio-Projekten, werden die Informationen über die neuen Dateien im Connectivity-Projekt hinterlegt.

Editor für Datenbankkonfigurationen

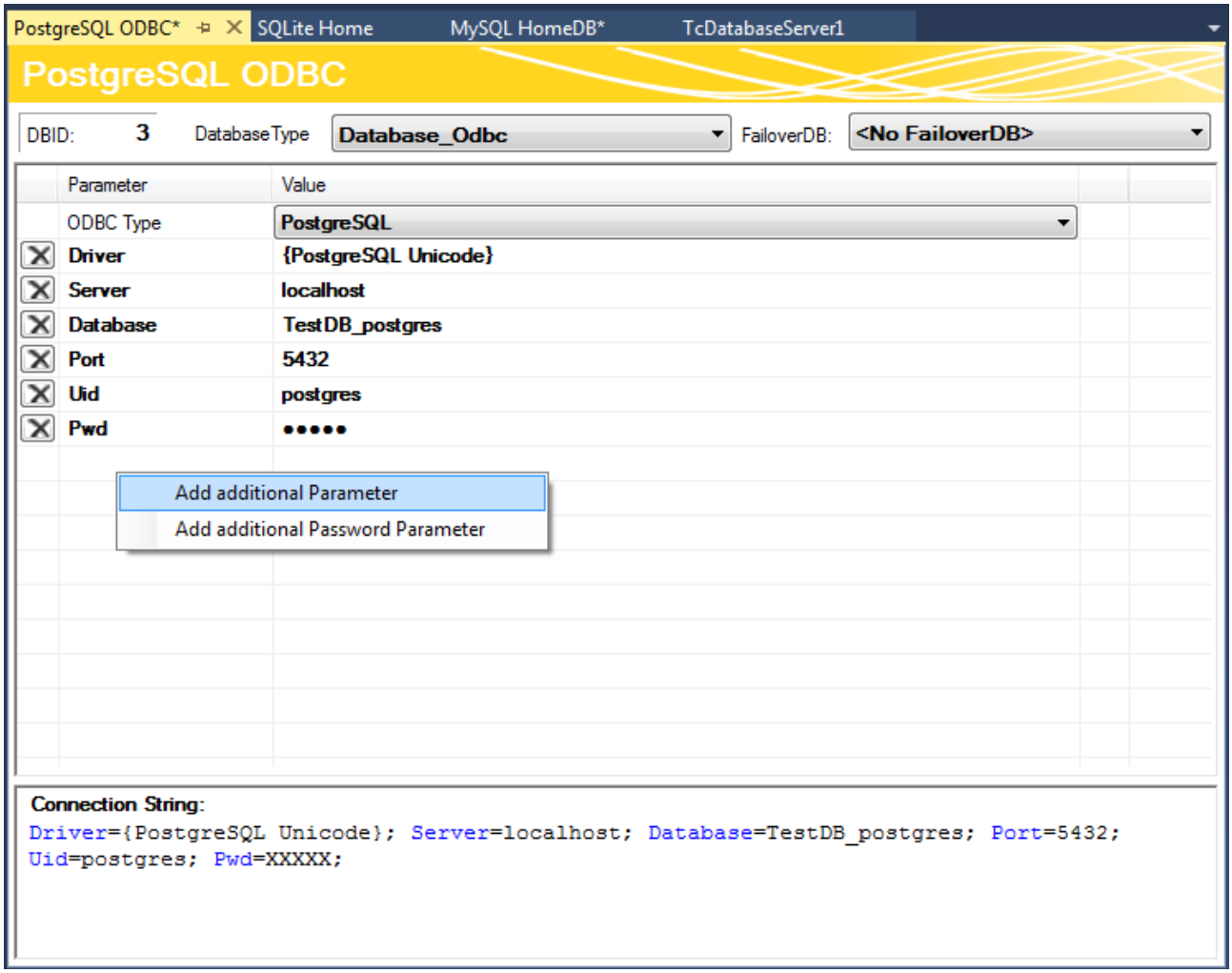


Im oberen Teil des Editors wird die Datenbank ID angezeigt (1), die für einige Funktionsbausteine in der SPS benötigt wird. Den Datenbanktyp der Zieldatenbank können Sie aus dem Drop-down-Menü auswählen (2). Hier können Sie auch die Odbc-Schnittstelle für eine Datenbank nutzen, die noch nicht unterstützt wird. Beachten Sie, dass je nach Datenbank nicht alle Funktionen des TwinCAT Database Servers gewährleistet werden.

Außerdem können Sie optional eine sogenannte FailOver-Datenbank (3) auswählen, welche im ‚Configure'-Mode beim Fehlerfall einspringt. Bei einem Verbindungsabbruch zum Netzwerk kann in einem solchen Fall automatisch gewährleistet werden, dass keine Daten verloren gehen und an anderer Stelle gespeichert werden.

Für jede Datenbank [▶ 126] stehen zusätzlich weitere einstellbare Parameter zur Verfügung. Je nach Datenbank wird ein Connection String (5) erstellt, welcher die Verbindung zur Datenbank beschreibt. Diese Anzeige dient zur Transparenz Ihrer eingestellten Parameter.

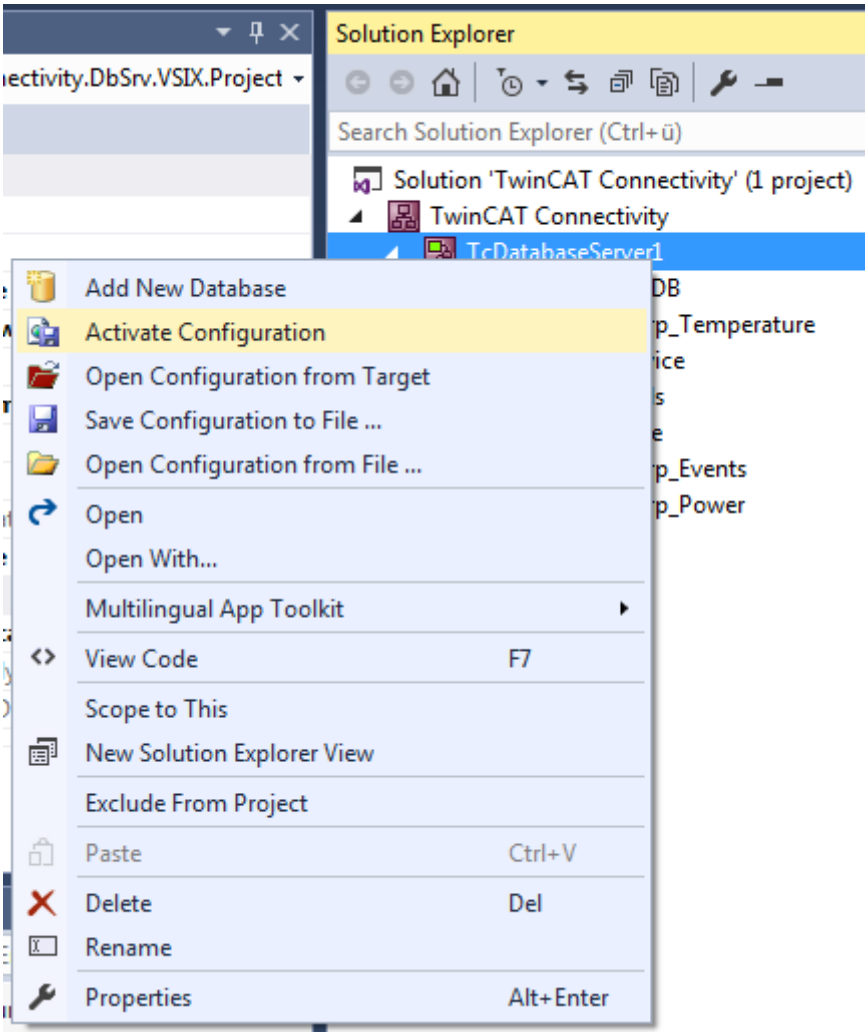
Über die Schaltfläche **CREATE** (4) können Sie eine neue Datenbank erstellen. Diese Funktion wird nur angezeigt, wenn sie von der jeweiligen Datenbank unterstützt wird.



Sie können auch unbekannte Datenbanken mit einer ODBC-Schnittstelle konfigurieren. Dafür wählen Sie im in der Drop-down-Liste **ODBC Type** den Eintrag „Unknown Database“ und fügen über die Befehle im Kontextmenü Parameter hinzu. Diese können auch Passwörter beinhalten, welche dann verschlüsselt abgespeichert werden. Daraus kann der gewünschte Connection String zusammengestellt werden. Beachten Sie, dass nur begrenzte Funktionen des TwinCAT Database Servers genutzt werden können. Nur die expliziten Funktionsbausteine des SQL Expert Modes werden unterstützt.

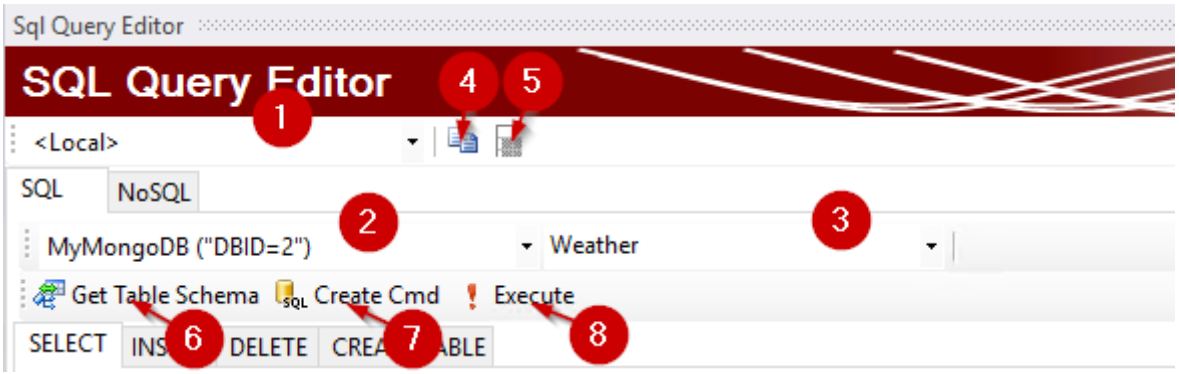
Projekt aktivieren

Um ein konfiguriertes Projekt auf dem TwinCAT Database Server zu aktivieren, verwenden Sie im Kontextmenü des TwinCAT-Database-Server-Projektes das Kommando **Activate Configuration**.



SQL Query Editor

Der SQL Query Editor ist ein Tool des Database Servers, um die Entwicklung Ihrer Applikation zu unterstützen. Mit dem Tool können Verbindungen und SQL-Befehle getestet und die Kompatibilität zwischen SPS und Datenbanken geprüft werden.



ID	Bezeichnung	Funktion
1	Zielsystem	Auswahl des Zielsystems mit installiertem TwinCAT Database Server
2	Datenbank	Auswahl der konfigurierten Datenbankverbindung
3	Tabelle	Auswahl der vorhandenen Tabellen in der Datenbank
4	Kopieren für SPS	Kopieren des SQL-Befehls in den SPS-String. Dieser kann in den SPS-Quellcode kopiert werden. Die Sonderzeichen werden automatisch erfasst und formatiert.
5	Export TC3	Exportieren des Tabellen-Schemas in eine SPS-Struktur. Diese kann beispielsweise im Programm für SQL-Befehle verwendet werden.
6	Get Table Schema	Auslesen der Tabellenstruktur
7	Create Cmd	Erstellen eines SQL-Befehls, basierend auf der Tabellenstruktur
8	Execute	Ausführen des SQL-Befehls

Wählen Sie dazu zunächst das Zielsystem aus den Routen Ihres TwinCAT-Systems aus (1). Der TwinCAT Database Server muss dafür auf dem Zielsystem installiert sein. Falls in der Konfiguration eine NoSql-Datenbank gespeichert sein sollte, ist zusätzlich noch ein NoSQL-Tab sichtbar. Die Dokumentation dazu finden Sie in einem späteren Unterpunkt.

Wenn Sie die Datenbankkonfigurationen auf dem Zielsystem aktiviert haben, werden alle konfigurierten Datenbanken (2) angezeigt. Weiterhin können Sie eine der verfügbaren Tabellen (3) von der Datenbank auswählen. Bezogen auf diese Tabelle können Sie einen SQL-Befehl aus dem SQL Query Editor heraus erzeugen und zur Datenbank senden. Je nach Datenbanktyp haben die SQL-Befehle eine unterschiedliche Syntax.

Um die einzelnen SQL-Befehle zu erzeugen, stehen Ihnen drei Kommandos zur Verfügung:

- Get Table Schema: Ruft die Struktur der ausgewählten Tabelle ab.
  - Diverse Informationen, wie Spaltennamen, PLC Datentyp, Größe der Variablen werden angezeigt. Die abgerufene Struktur können Sie außerdem über die Befehle **Kopieren für SPS** (4) oder **Export TC3** (5) für Ihre SPS-Applikation aufbereiten.
- Create Cmd: Abhängig von der ausgewählten Registerkarte, wird ein SQL-Befehl in der Befehls-Textbox erzeugt. Die Syntax des Kommandos kann je nach Datenbanktyp unterschiedlich sein. Hier wird das zuvor ausgelesene Tabellenschema benutzt.
  - Der erzeugte SQL-Befehl kann optional auch verändert werden.
- Execute: Der SQL-Befehl, der in der Textbox hinterlegt ist, wird ausgeführt und liefert gegebenenfalls Werte zurück.

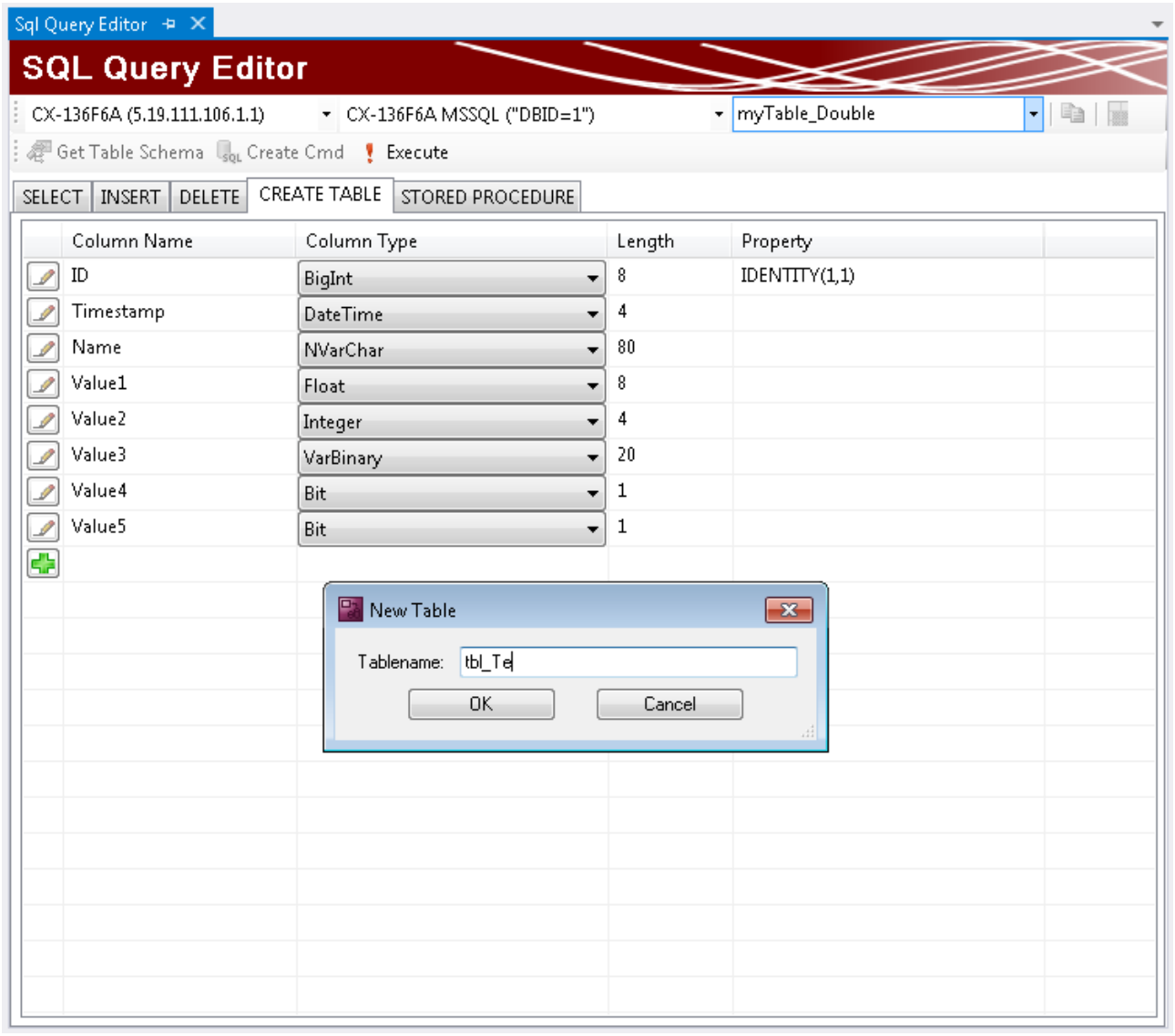
Auf die Unterschiede der einzelnen SQL-Befehle wird im Folgenden separat eingegangen.

Bemerkung: Da die Syntax der SQL-Befehle oft mit der Syntax im ST-Code von TwinCAT kollidiert, bietet der SQL Query Editor das Kommando "Kopieren für SPS" (4). Mit dem Kommando werden die erzeugten und getesteten SQL-Befehle mit der richtigen Formatierung für Sonderzeichen für den ST-Programmcode in den Zwischenspeicher kopiert.

Create-Table-Befehl

In der Registerkarte **CREATE TABLE** können Tabellen innerhalb der Datenbank erzeugt werden. Je nach Anforderung können Sie der Tabelle weitere Spalten hinzufügen (+). Nachdem Sie Spaltenname und -typ festgelegt haben, können Sie zusätzliche Properties angegeben, um zum Beispiel automatisch generierte IDs zu erzeugen.

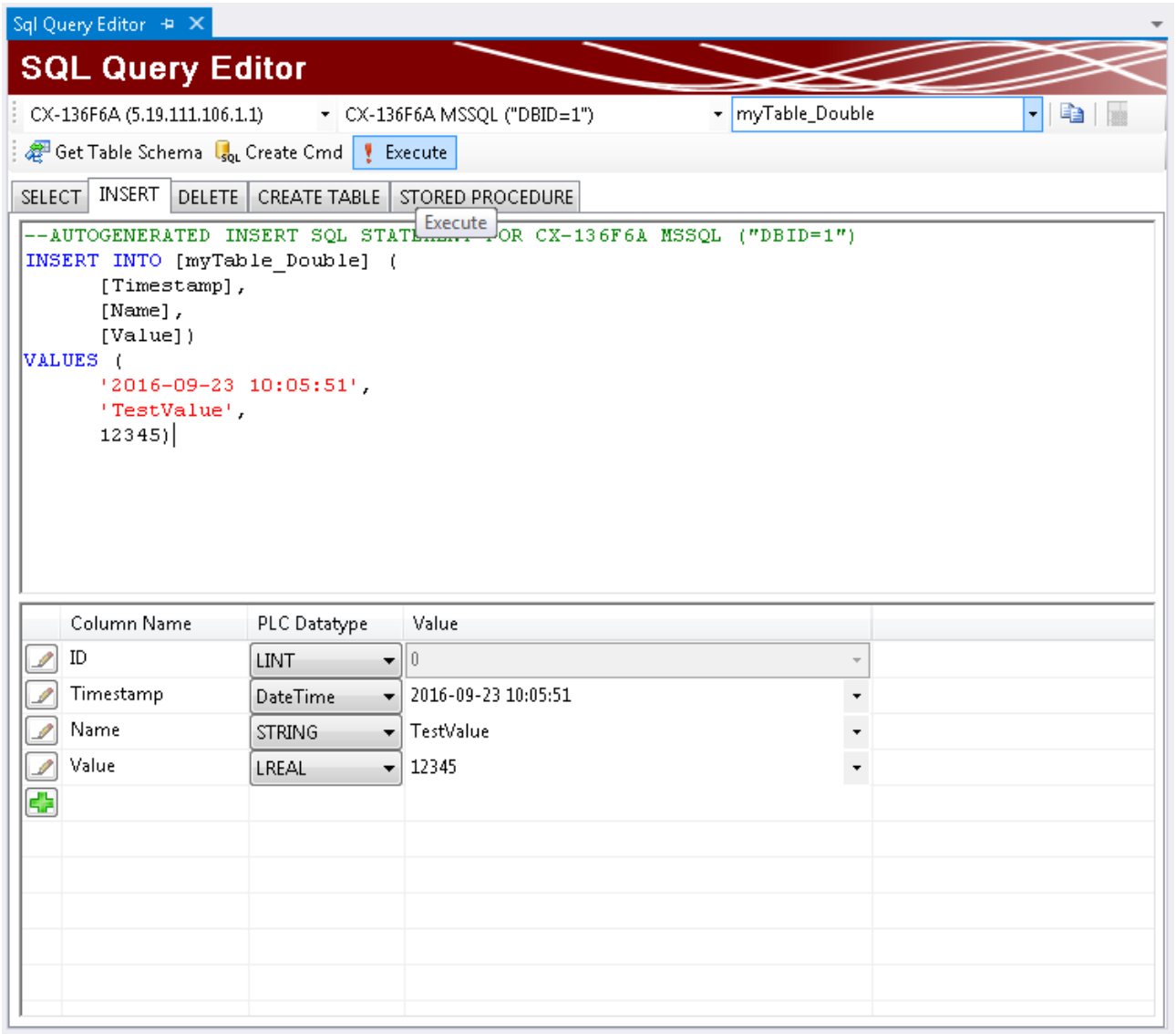
Durch das Ausführen des Kommandos kann der Tabellenname bestimmt werden. Die Tabelle mit der konfigurierten Tabellenstruktur wird erstellt.



Insert-Befehl

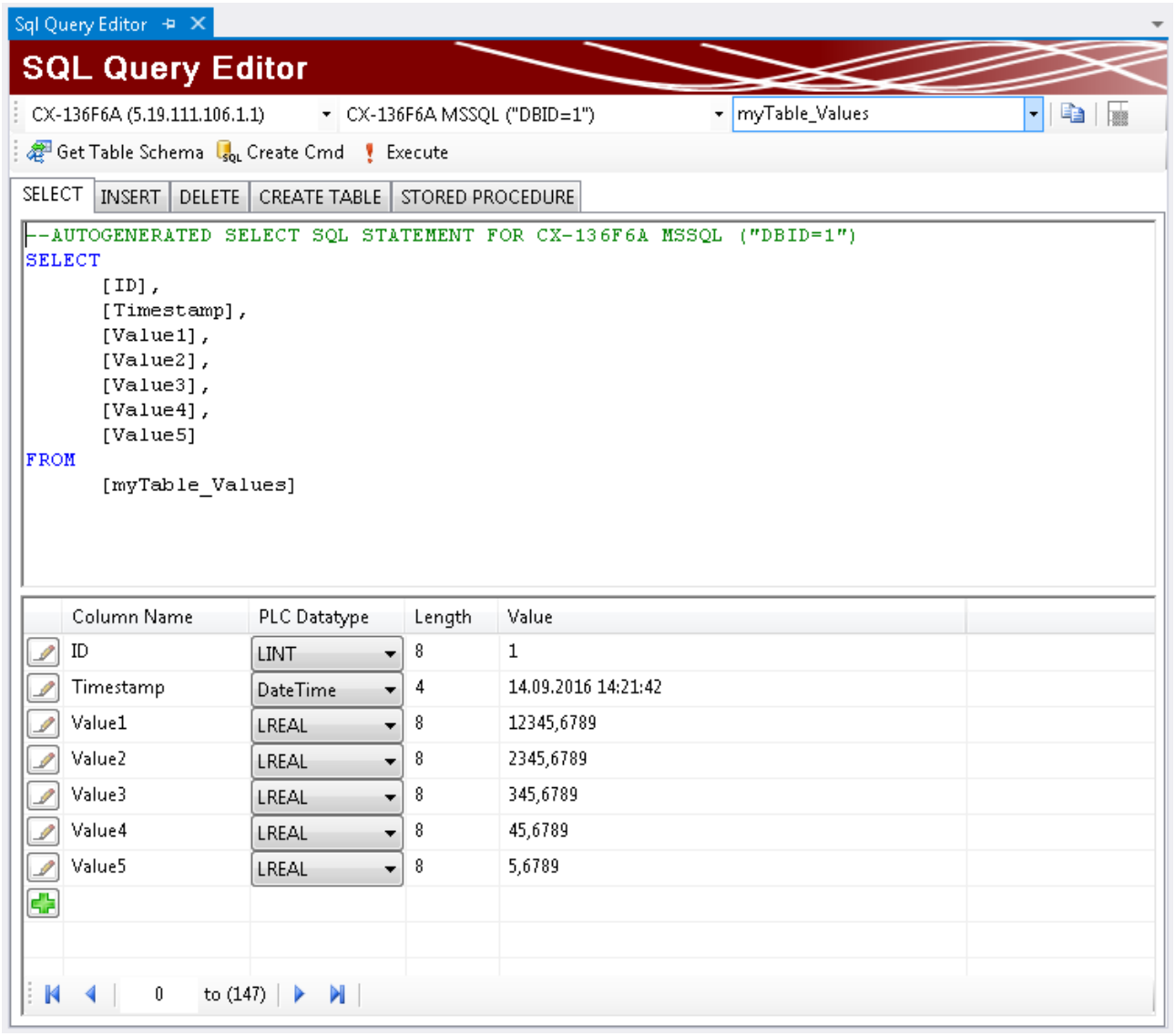
Der Insert-Befehl gibt die Möglichkeit, Datensätze in die Tabelle zu schreiben. Nachdem die Tabellenstruktur abgeholt wurde, können die Werte unter „Value“ verändert werden. Wenn daraufhin das Kommando erzeugt wird, stehen diese Werte automatisch im Insert-Befehl im richtigen Format. Durch das Ausführen des Befehls werden diese Werte in die Tabelle geschrieben.

Wenn eine automatische ID-Erzeugung verwendet wird, kann dieser Wert nicht angepasst werden.



Select-Befehl

In der Registerkarte **SELECT** können Select-Befehle erstellt und abgesendet werden. Select-Befehle ermöglichen es, Datensätze aus Datenbanken abzurufen. Nach dem Ausführen des Befehls werden Werte zurückgeliefert, wenn diese in der Tabelle vorhanden sind. In der Anzeige der Tabellenstruktur werden sie unter „Value“ gelistet. Die Pfeile unter der Anzeige dienen zur Navigation durch die einzelnen Datensätze.

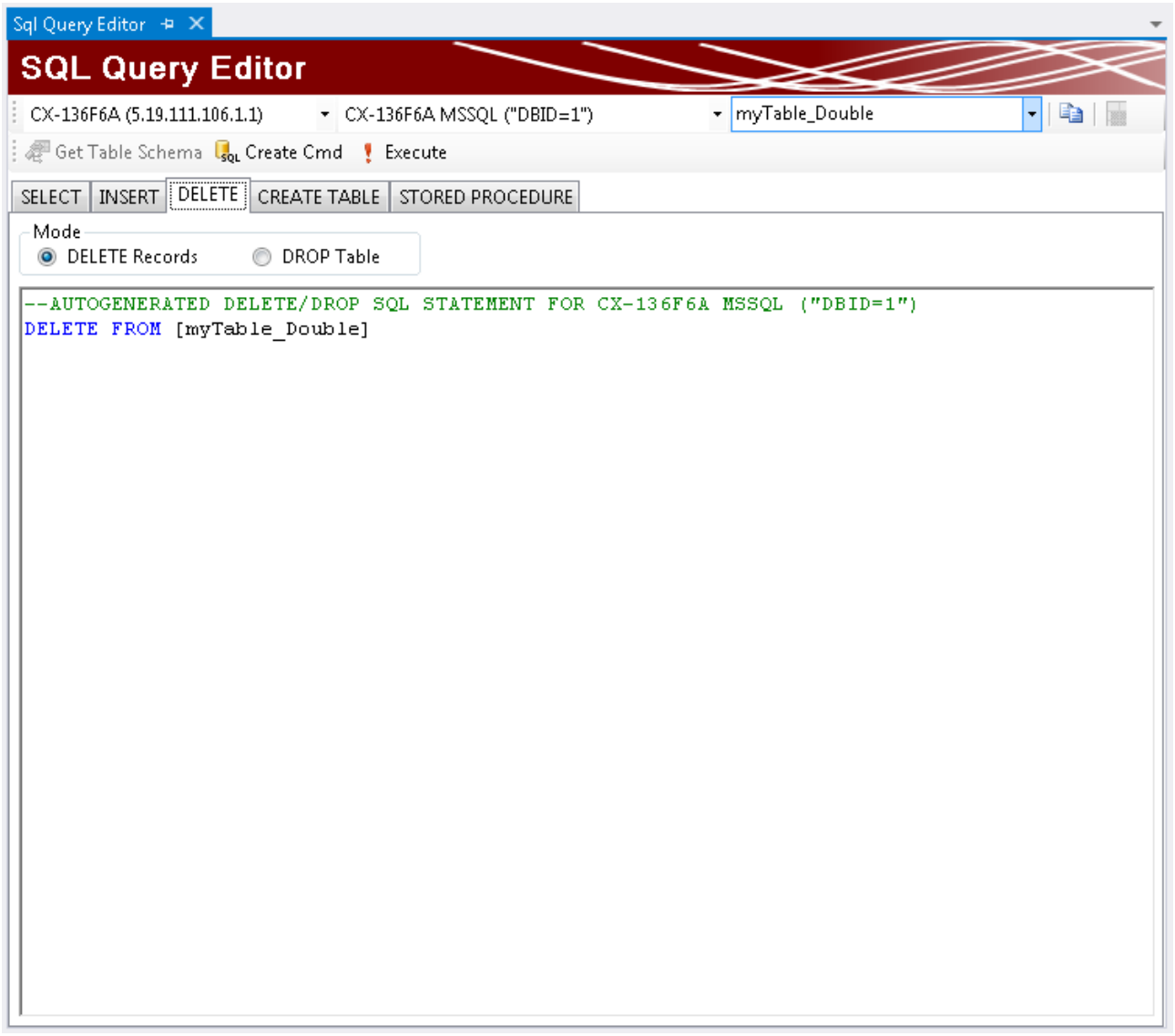


Delete-Befehl

Der Delete-Befehl verfügt über zwei Funktionsweisen.

1. **DELETE Records**: Löscht den Inhalt einer Tabelle.
2. **DROP Table**: Löscht die gesamte Tabelle.

Auch diesen SQL-Befehl können Sie vor dem Senden anpassen, um z. B. nur einen bestimmten Bereich der Tabelle zu löschen.

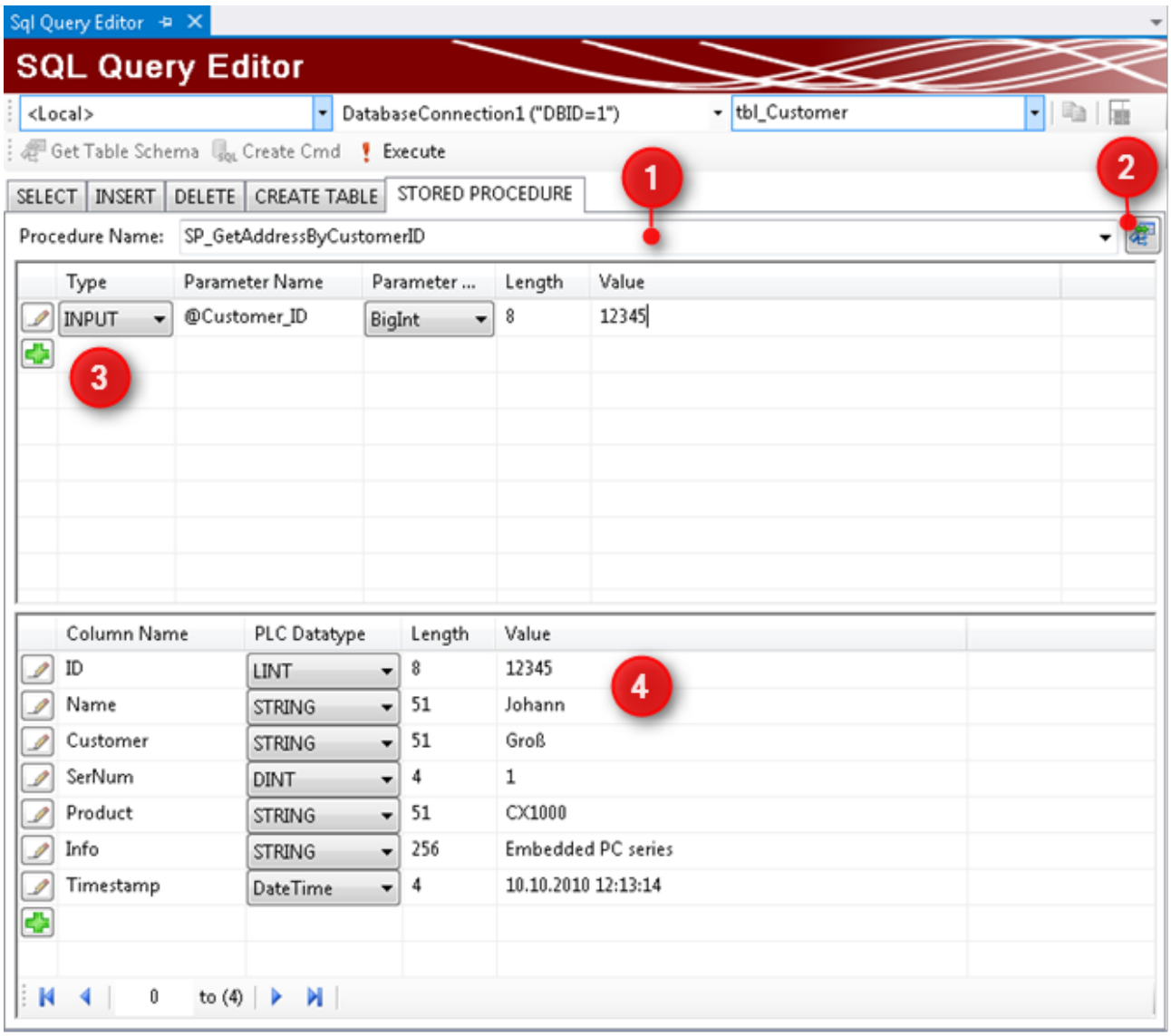


Stored Procedures

Der TwinCAT Database Server unterstützt „Stored Procedures“, die viele Datenbanken bereitstellen, um komplexere Abfragen auf der Datenbankebene zu verarbeiten oder eine vereinfachte Schnittstelle zur Verfügung zu stellen.

Falls Stored Procedures in der Datenbank und Tabelle vorhanden sind, können Sie diese erfassen und auswählen (1). Die Ein- und Ausgabeparameter können Sie automatisch aufnehmen (2) und in die Tabellen in der Anzeige übertragen (3)(4).

Dort werden der Parametertyp, Name und Datentyp angezeigt. Zusätzlich können Sie hier Werte einfügen, um die Stored Procedure mit den Eingabewerten mit „Execute“ auszuführen. Das Ergebnis wird daraufhin in den Ausgabewerten angezeigt (4). Bei mehreren zurückgelieferten Datensätzen ist ein Durchschalten mit den Pfeiltasten möglich. Diese Funktionalität dient als Entwicklungshilfe für den Aufruf in der SPS. Dort werden diese Ergebnisse bei einem Aufruf des entsprechenden Funktionsbausteins [▶ 205] zurückgeliefert.

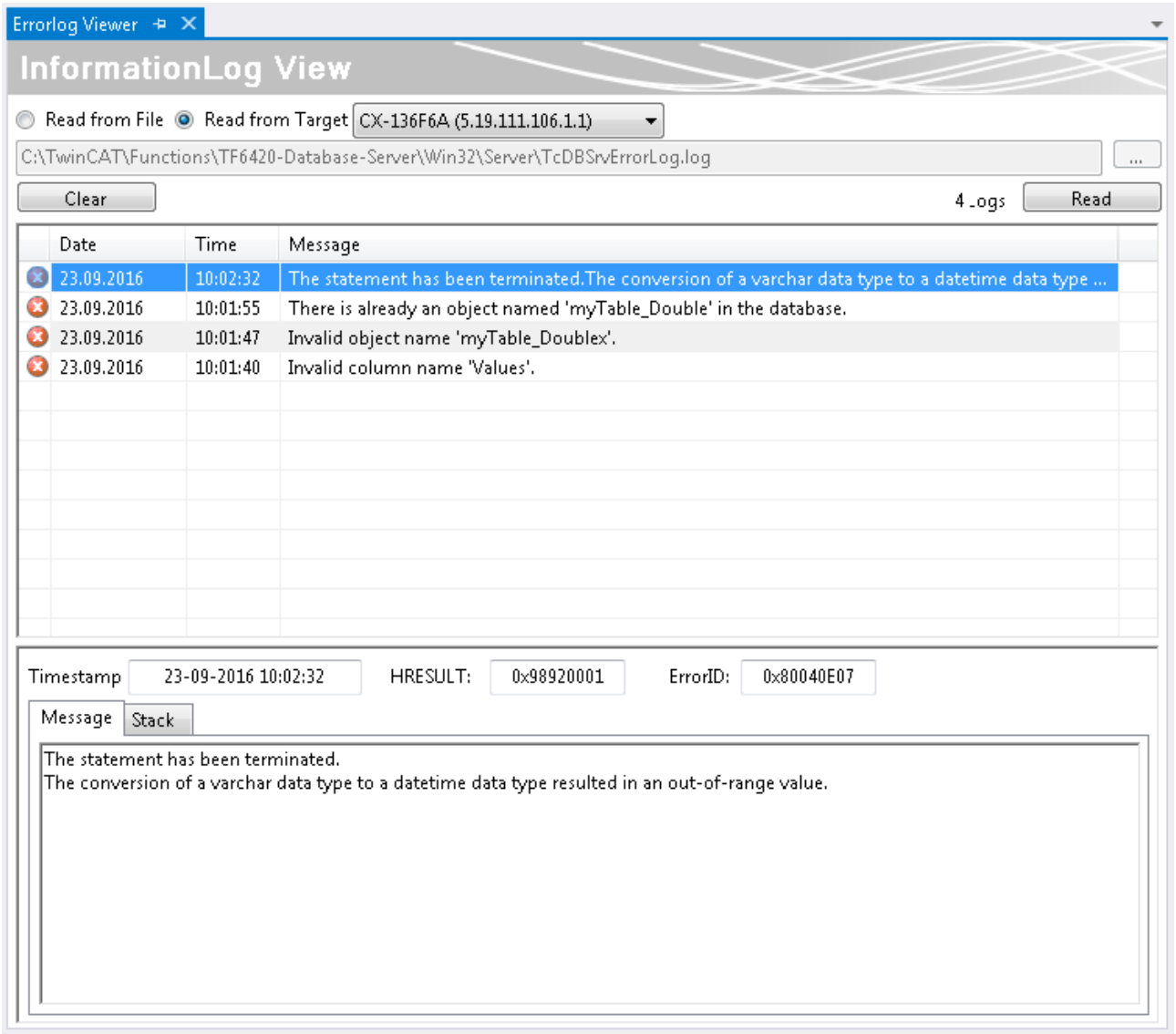


InformationLog View zur Diagnose

Um fehlerhaftes Verhalten zu diagnostizieren, steht der InformationLog View zur Verfügung.

Der InformationLog View ist ein Tool, um die Logdateien vom TwinCAT Database Server auszulesen. Protokollierte Informationen werden mit einem Zeitstempel, IDs und Fehlermeldungen im Klartext angezeigt.

Die Log-Dateien können nicht nur über den direkten Dateizugriff eingesehen oder geleert werden, sondern auch direkt über das Target. Gerade für verteilte Database Server im Netzwerk ist dies vorteilhaft, um einen schnellen und einfachen Zugriff auf die Logdatei zu erlangen. Für diesen Zugriff muss eine Route zum Zielgerät bestehen.



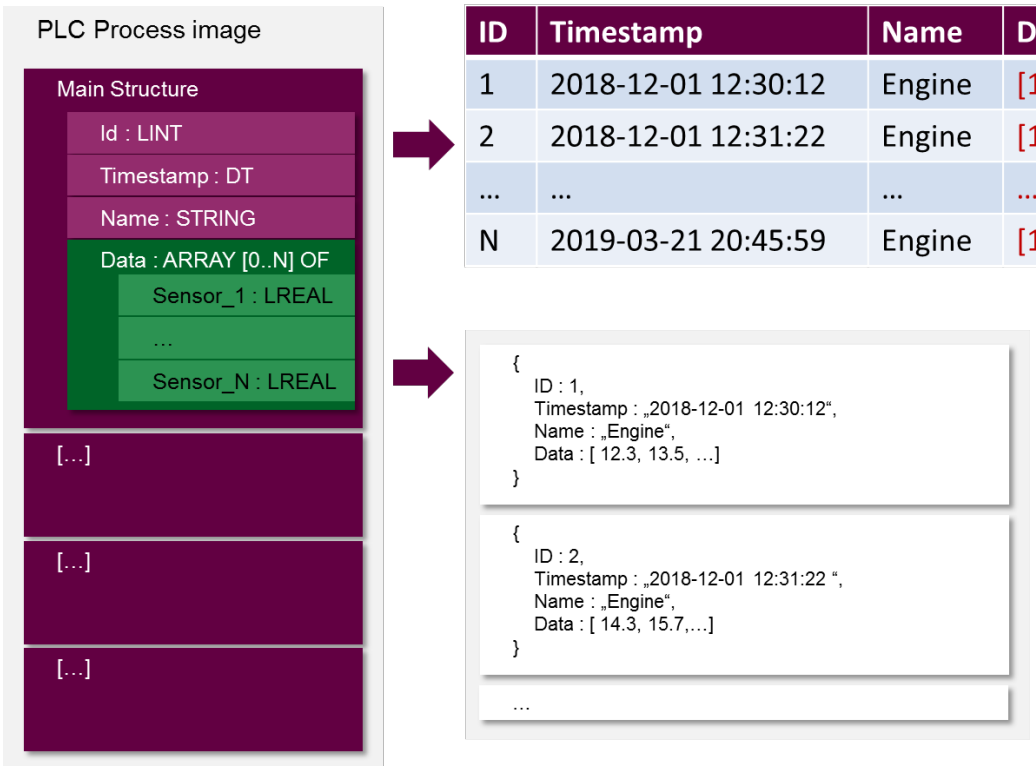
5.1.1.5 NoSql Expert Mode

NoSQL

NoSql-Datenbanken (not only Sequel) sind Datenbanken, die von der herkömmlichen relationalen Datenspeicherung abweichen.

Dokumentbasierte Datenbanken:

Datensätze werden als Dokumente in der Datenbank abgelegt. Das bietet den Vorteil flexibler und auch hierarchisch tiefergehend Daten zu archivieren.



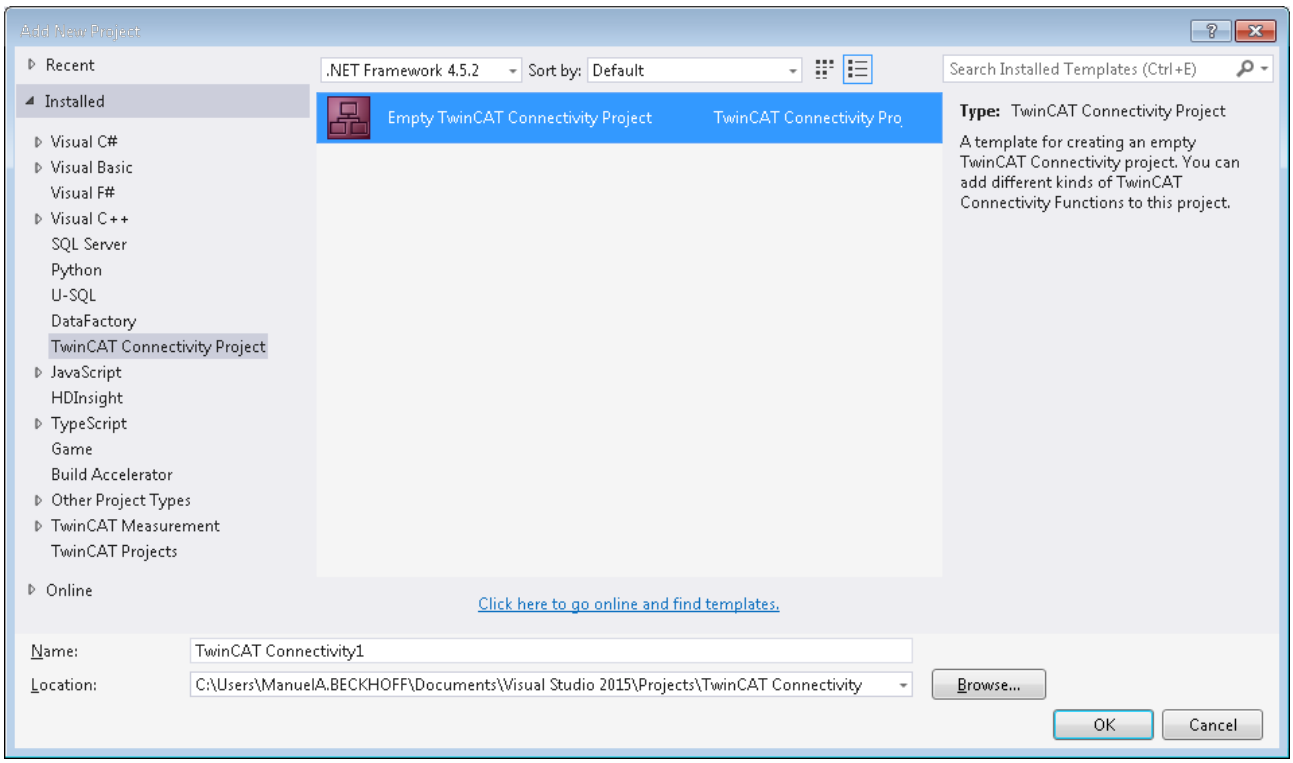
Falls ein Datensatz aus mehr als einer flachen Struktur von Basisdatentypen besteht, kann diese nicht mehr ohne Umwege über eine relationale Datenbank abgebildet werden. NoSql-Datenbanken bieten diese Flexibilität. Auch Veränderungen im Programmcode und den entsprechenden Strukturen können einfach übernommen werden, ohne dass eine neue Tabelle dafür angelegt werden muss.

Dokumentbasierte Datenbanken speichern die Daten meist als Json-formatierte Datensätze. Jeder Datensatz kann dabei vom anderen abweichen.

Projekt erstellen

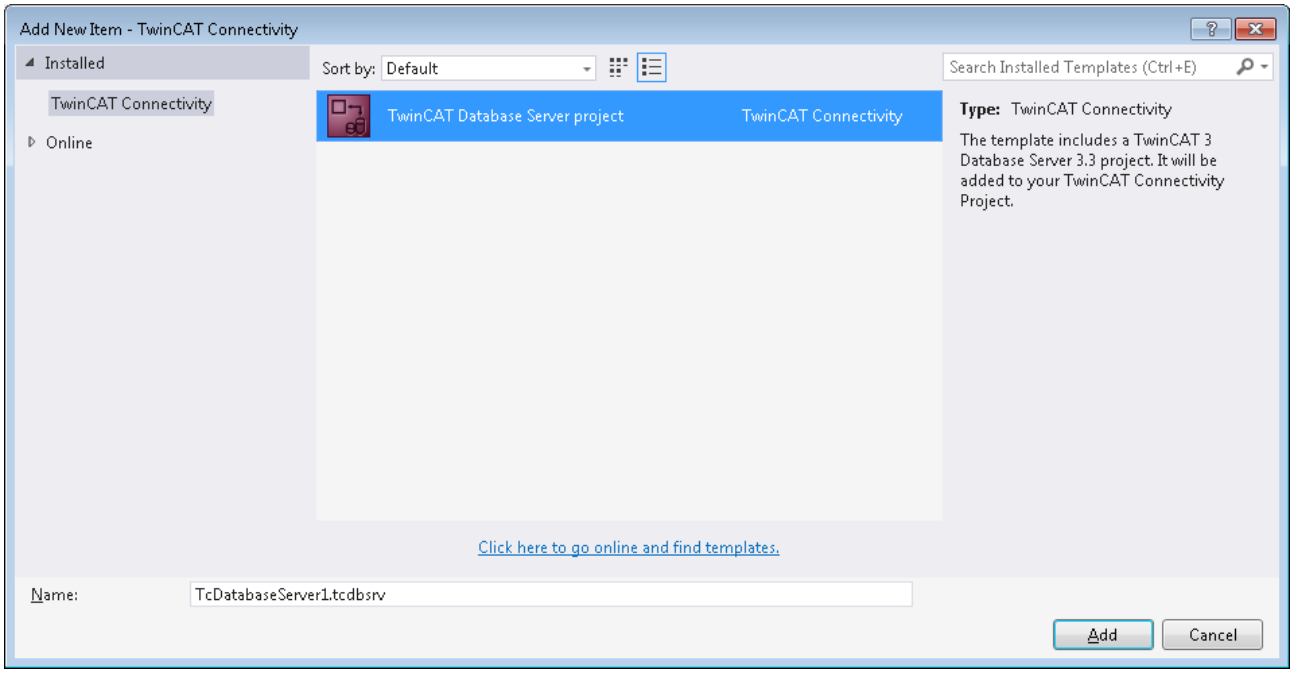
Durch die TwinCAT-Connectivity-Erweiterung für Visual Studio steht eine neue Projektvorlage zur Verfügung. Beim Erstellen eines neuen Projektes erscheint nun die Kategorie **TwinCAT Connectivity Project** in der Auswahl.

Um ein neues TwinCAT-Connectivity-Projekt zu erstellen, wählen Sie das **Empty TwinCAT Connectivity Project**, legen den Projektnamen und den Speicherort fest und fügen es mit **OK** der Solution hinzu. TwinCAT-Connectivity-Projekte bzw. TwinCAT-Database-Server-Projekte können so komfortabel neben TwinCAT- oder anderen Visual-Studio-Projekten angelegt werden.

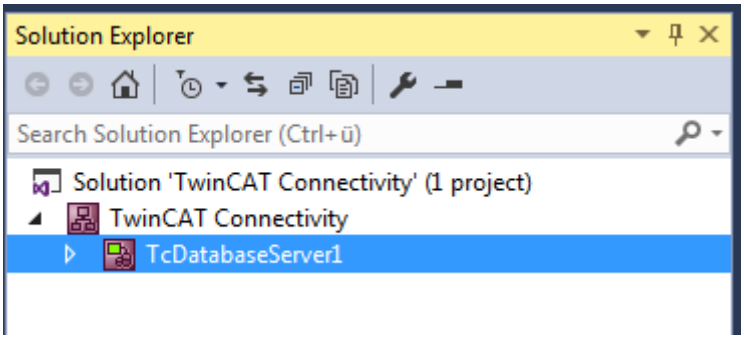


In der Solution erscheint ein neuer Projektknoten. Unterhalb des Connectivity-Projektknotens können Sie Subprojekte der unterstützten Connectivity-Funktionen ergänzen.

Mit **Add** können Sie dem TwinCAT-Connectivity-Projekt ein neues TwinCAT-Database-Server-Projekt hinzufügen. Das TwinCAT-Database-Server-Projekt befindet sich in der Auflistung der vorhandenen Item Templates.



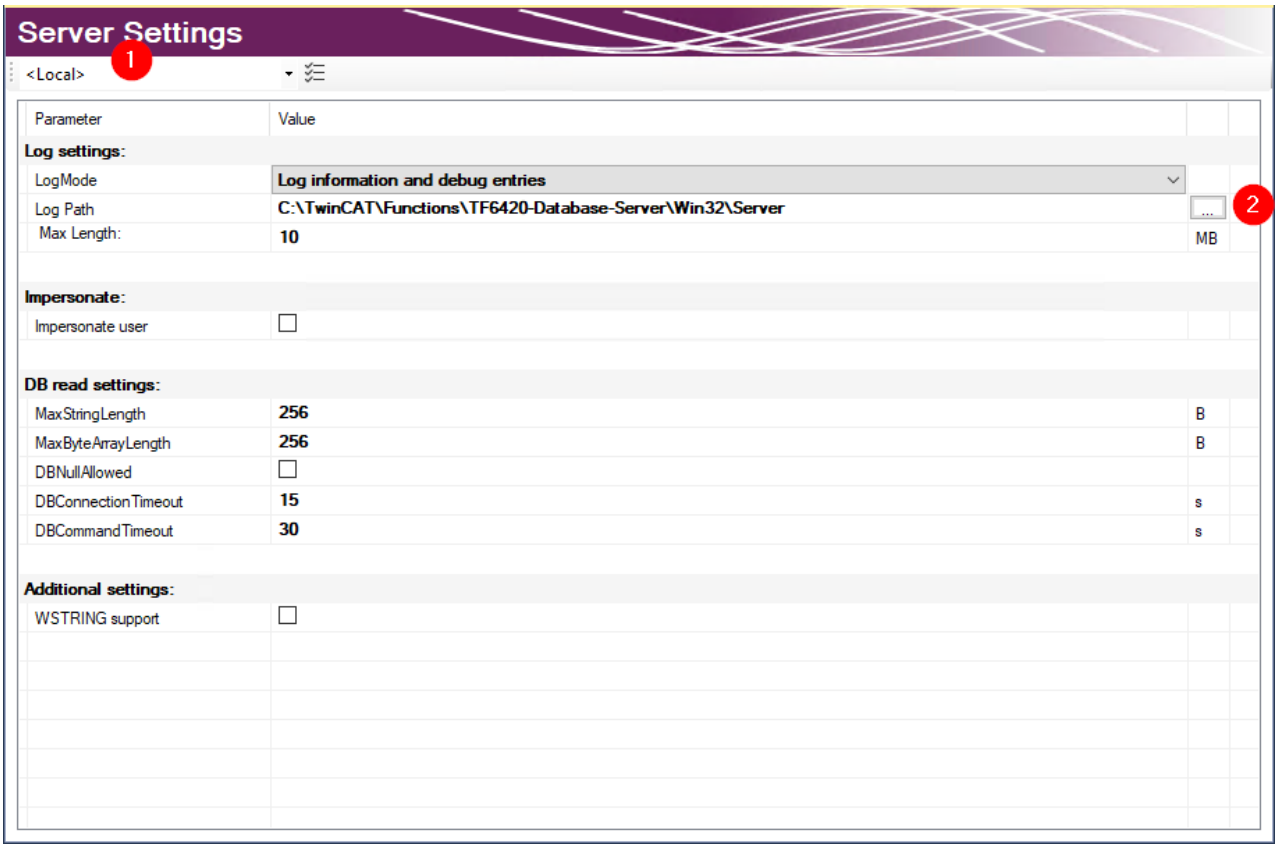
Unterhalb des TwinCAT-Connectivity-Knotens wird ein neues TwinCAT-Database-Server-Projekt angelegt.



Dieses dient nun als Basis für die anstehende Konfiguration eines TwinCAT Database Servers. Das Dokument können Sie sowohl über die Eigenschaften im Eigenschaftsfenster, als auch über einen Editor bearbeiten.

Einem Connectivity-Projekt können Sie beliebig viele TwinCAT-Database-Server-Projekte oder andere Projekte hinzufügen, und damit auch mehrere Konfigurationen in einem Connectivity-Projekt einstellen.

Editor für Server-Einstellungen



Im Editor **Server Settings** können Sie die Einstellungen für den TwinCAT Database Server bearbeiten. Diese Einstellungen betreffen den entsprechenden Server im Allgemeinen. Im Drop-down-Menü (1) können Sie das Zielsystem über die Ams-NetId angegeben. Dafür müssen Sie über TwinCAT eine Route zum Zielsystem anlegen. Wird eine fertige Konfiguration übertragen, werden die Einstellungen auf den TwinCAT Database Server dieses Zielsystems gespeichert.

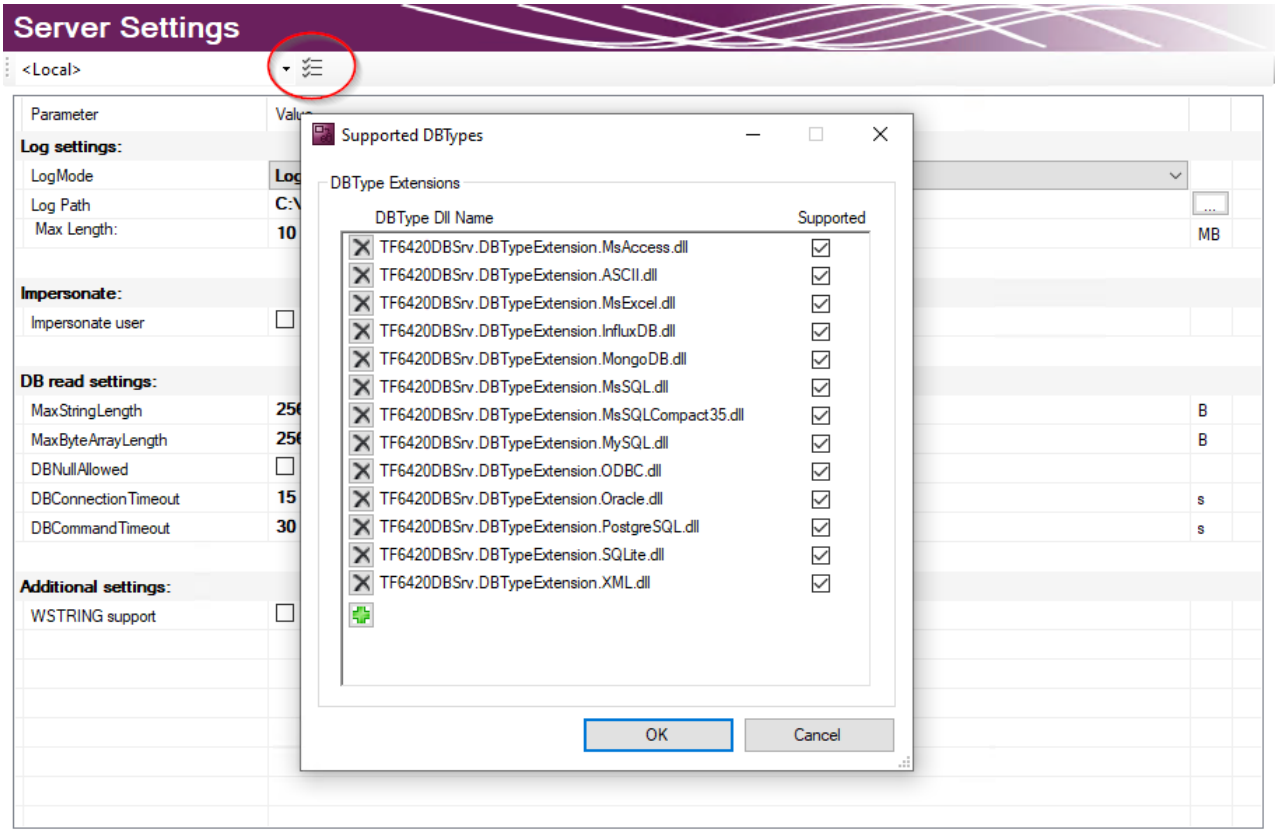
In den **Log settings** konfigurieren Sie Einstellungen zum Aufzeichnen von Fehlerfällen. In einem Fehlerfall erzeugt der Database Server einen detaillierten Eintrag in einer Logdatei. Diese können Sie mit dem Information Log Viewer [▶ 51] auslesen. In den **Log Settings** geben Sie einen Pfad zum Ablageort und die maximale Größe der Datei an. Zusätzlich können Sie die Genauigkeit des Logs beeinflussen. Wir empfehlen, aus Performancegründen, das Loggen nach erfolgter Fehleranalyse wieder abzuschalten, wenn es nicht mehr benötigt wird.

Bei Netzwerkzugriff auf dateibasierende Datenbanken wie Access Datenbanken oder SQL Compact Datenbanken müssen Sie die Option **Impersonate** setzen, damit sich der TwinCAT Database Server mit dem Netzwerklaufwerk verbinden kann. **Diese Funktion wird zurzeit nicht unter Windows CE unterstützt.**

Sie können weitere Einstellungen konfigurieren, um das Lesen aus der Datenbank zu steuern. Diese Einstellungen beziehen sich auf den TwinCAT Database Server auf dem Zielsystem:

MaxStringLength	Maximale String-Länge der Variablen in der SPS
MaxByteArrayLength	Maximale Byte-Array-Länge der Variablen in der SPS
DBNullAllowed	Gibt an, ob NULL-Werte im TwinCAT Database Server angenommen werden.
DBConnectionTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem Verbindungsaufbau von einem Verbindungsfehler ausgeht.
DBCommandTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem abgesendeten Kommando von einem Verbindungsfehler ausgeht. Bei hohen Datenmengen kann, je nach Datenbank und Infrastruktur ein Befehl durchaus sehr viel Zeit in Anspruch nehmen.

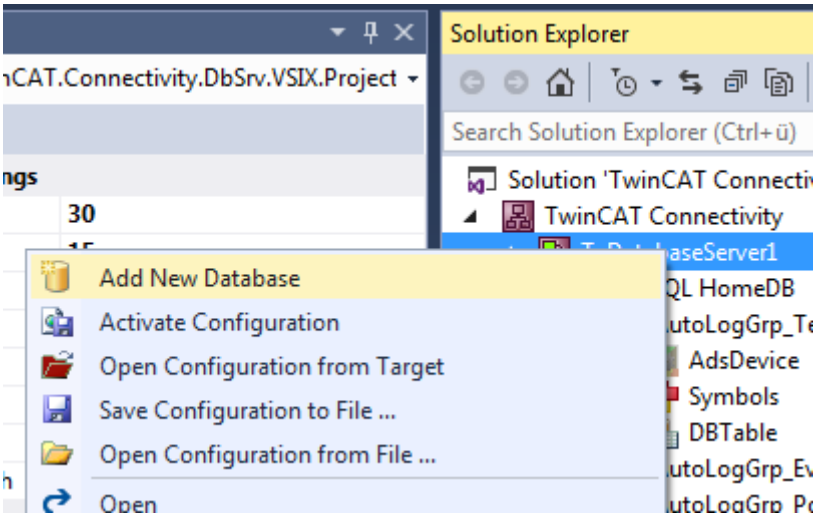
Unterstützte Datenbanktypen



Die installierten Datenbanktypen können in den Server Einstellungen ausgewählt werden. Standardmäßig sind alle installierten Datenbanken angewählt. Der TwinCAT 3 Database Server wird die entsprechenden Datenbankschnittstellen laden. Auf diese Weise können nicht genutzte Datenbanken auf dem Zielsystem abgewählt werden.

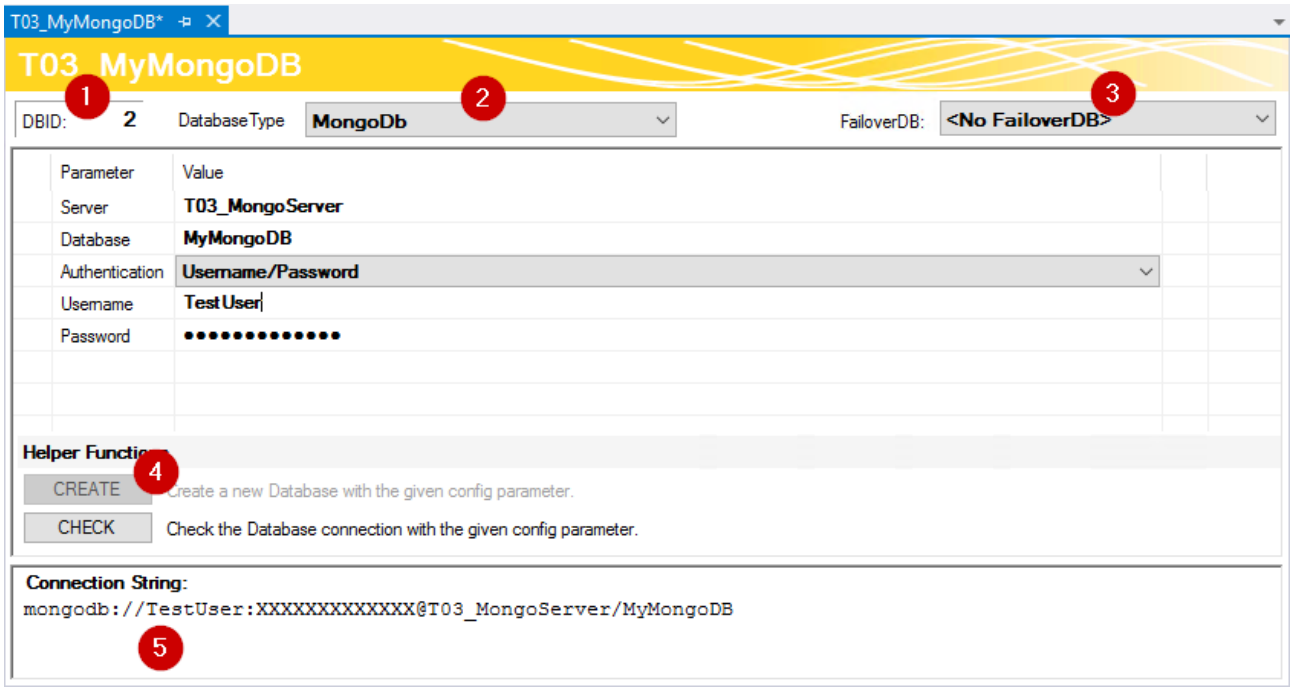
Datenbankkonfiguration hinzufügen

Eine neue Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new Database** über das Kontextmenü eines Database-Server-Projektes oder das entsprechende Kommando in der Toolbar hinzufügen.



Eine neue Datenbankkonfiguration wird als Datei auf im Projektordner hinzugefügt und in das Projekt eingebunden. Wie bei allen Visual-Studio-Projekten, werden die Informationen über die neuen Dateien im Connectivity-Projekt hinterlegt.

Editor für Datenbankkonfigurationen



Im oberen Teil des Editors wird die Datenbank ID angezeigt (1), die für einige Funktionsbausteine in der SPS benötigt wird. Den Datenbanktyp der Zieldatenbank können Sie aus dem Drop-down-Menü auswählen (2). Hier können Sie auch die Odbc-Schnittstelle für eine Datenbank nutzen, die noch nicht unterstützt wird. Beachten Sie, dass je nach Datenbank nicht alle Funktionen des TwinCAT Database Servers gewährleistet werden.

Außerdem können Sie optional eine sogenannte FailOver-Datenbank (3) auswählen, welche im ‚Configure'-Mode beim Fehlerfall einspringt. Bei einem Verbindungsabbruch zum Netzwerk kann in einem solchen Fall automatisch gewährleistet werden, dass keine Daten verloren gehen und an anderer Stelle gespeichert werden.

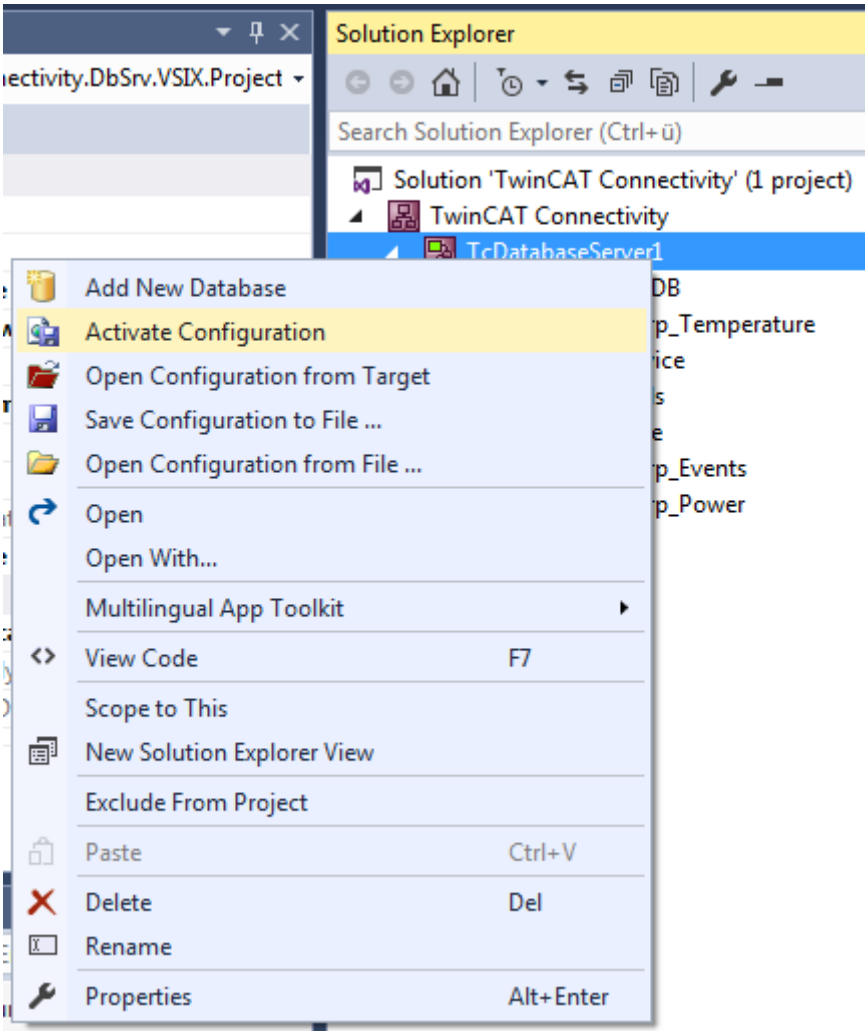
Für jede Datenbank [▶ 126] stehen zusätzlich weitere einstellbare Parameter zur Verfügung. Je nach Datenbank wird ein Connection String (5) erstellt, welcher die Verbindung zur Datenbank beschreibt. Diese Anzeige dient zur Transparenz Ihrer eingestellten Parameter.

Über die Schaltfläche **CREATE** (4) können Sie eine neue Datenbank zu erstellen. Diese Funktion wird nur angezeigt, wenn sie von der jeweiligen Datenbank unterstützt wird. Mithilfe von **CHECK** kann die Verbindung zur Datenbank überprüft werden.

Unter den Zieldatenbanken können auch NoSQL-Datenbanken ausgewählt werden. Aktivieren Sie das Projekt mit einer NoSQL-Datenbank, wird im SQL-Query Editor ein NoSql-Tab freigeschaltet, um NoSql-spezifische Funktionen nutzen zu können.

Projekt aktivieren

Um ein konfiguriertes Projekt auf dem TwinCAT Database Server zu aktivieren, verwenden Sie im Kontextmenü des TwinCAT-Database-Server-Projektes das Kommando **Activate Configuration**.



MongoDB im PLC Expert/Configure Mode

Der PLC Expert und der Configure Mode verwenden in ihren Abläufen das vorgegebene Schema einer Datenbank. Im Normalfall wird sich das Schema der verwendeten Strukturen im laufenden Betrieb nicht ändern. Um dennoch die Bausteine nutzen zu können, benötigt der TwinCAT 3 Database Server eine Beschreibung des Tabellenschemas. Für MongoDB wird deshalb eine Tabelle simuliert.
Verwenden Sie im SQL Query Editor den Reiter **SQL**, sowie die Unterkategorie **CREATE TABLE** um eine Tabelle bzw. in diesem Fall eine Collection zu erstellen. Zusätzlich wird, anders als bei relationalen Datenbanken, ein Eintrag in eine Metadaten-Collection erzeugt. Hier sind Informationen zum Tabellenschema für den TwinCAT 3 Database Server abgespeichert.

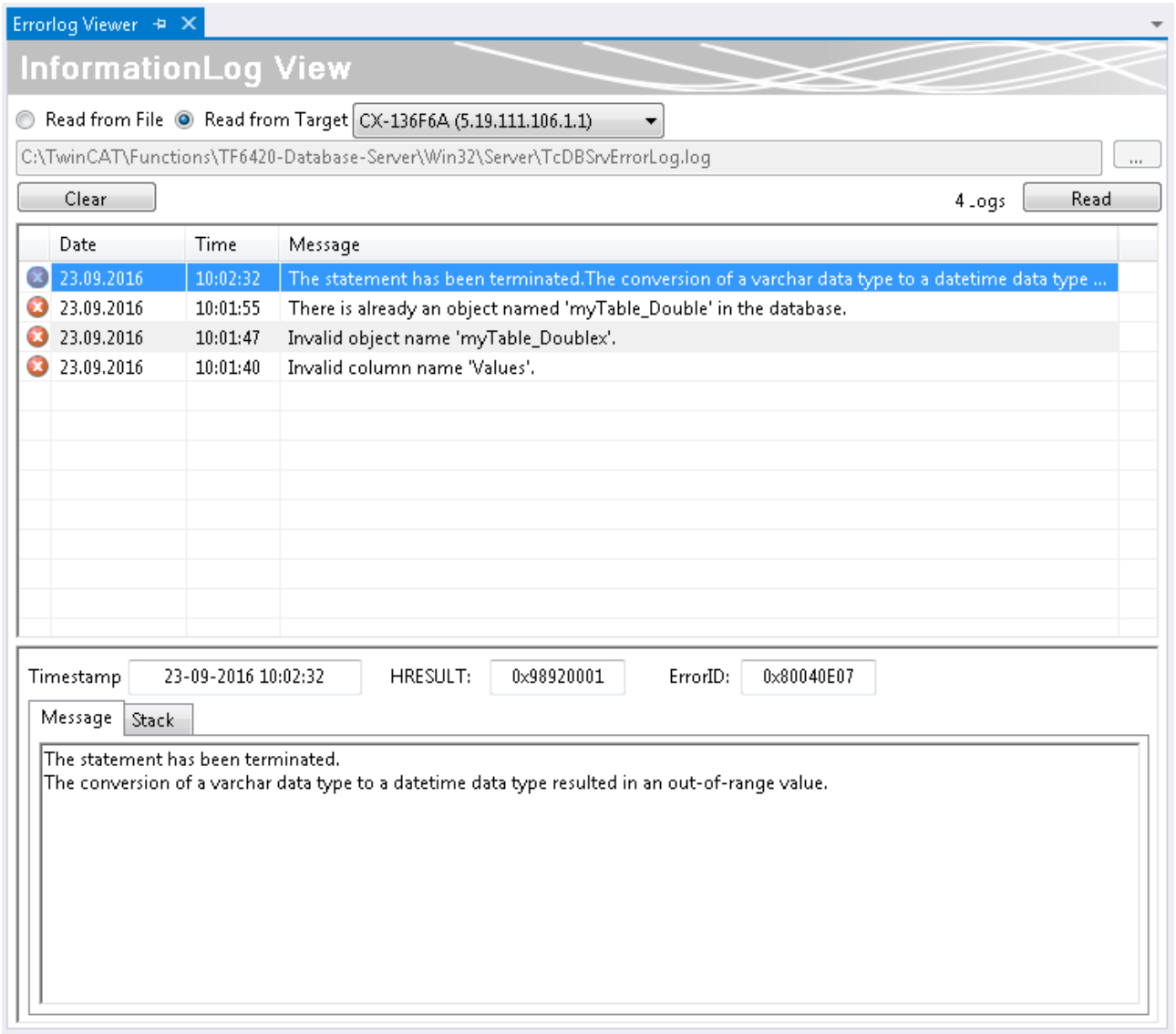
Um erweiterte Funktionsmöglichkeiten zu nutzen, wie z.B. Strukturen beliebiger Hierarchie oder flexible Datensätze, wird empfohlen die NoSQL-Funktionsbausteine zu verwenden.

InformationLog View zur Diagnose

Um fehlerhaftes Verhalten zu diagnostizieren, steht der InformationLog View zur Verfügung.

Der InformationLog View ist ein Tool, um die Logdateien vom TwinCAT Database Server auszulesen. Protokollierte Informationen werden mit einem Zeitstempel, IDs und Fehlermeldungen im Klartext angezeigt.

Die Log-Dateien können nicht nur über den direkten Dateizugriff eingesehen oder geleert werden, sondern auch direkt über das Target. Gerade für verteilte Database Server im Netzwerk ist dies vorteilhaft, um einen schnellen und einfachen Zugriff auf die Logdatei zu erlangen. Für diesen Zugriff muss eine Route zum Zielgerät bestehen.

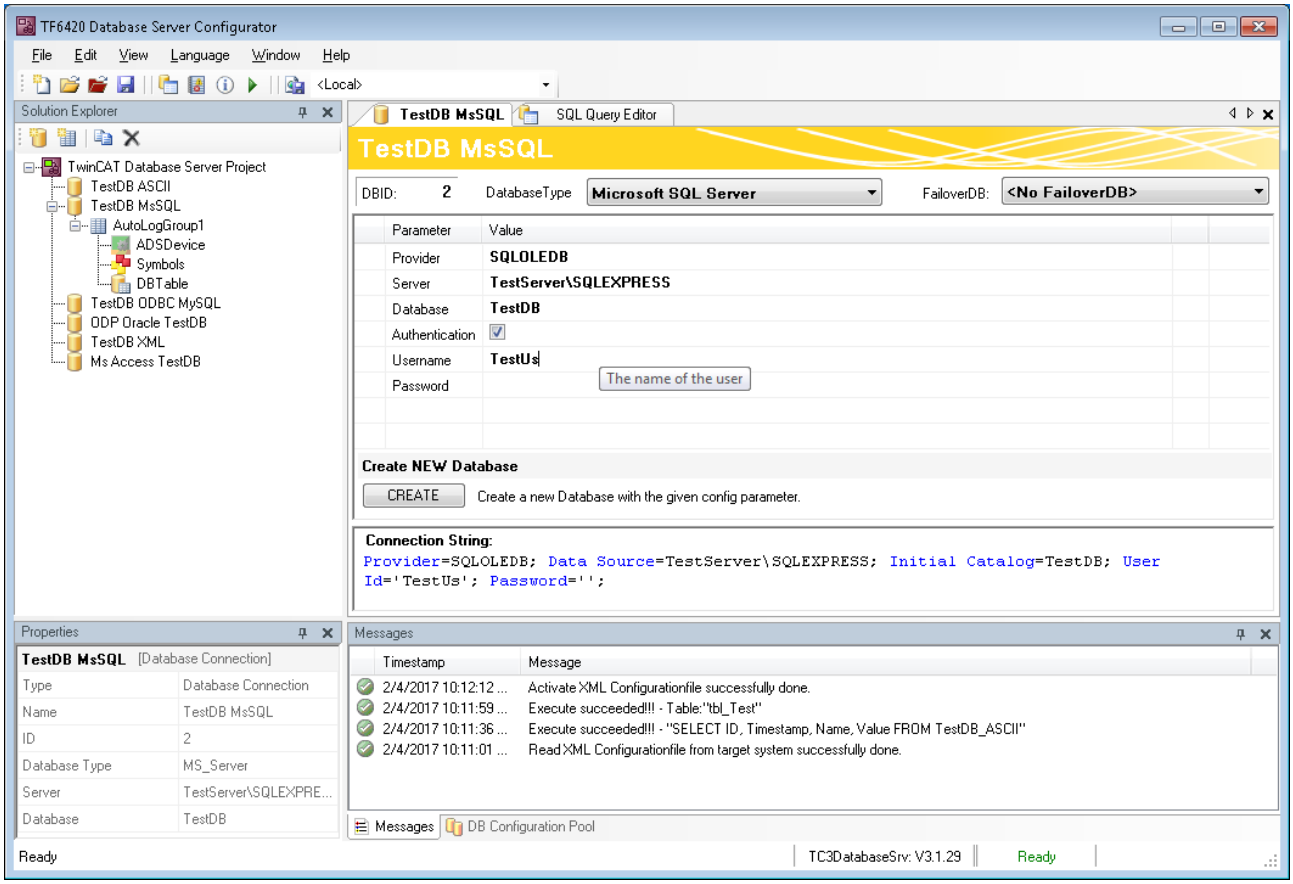


5.1.2 Standalone-Konfigurator

5.1.2.1 Allgemein

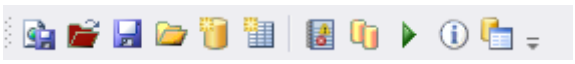
5.1.2.1.1 Oberfläche und grundlegende Funktionen

Die Konfiguration des TwinCAT 3 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.



Toolbar und Kommandos

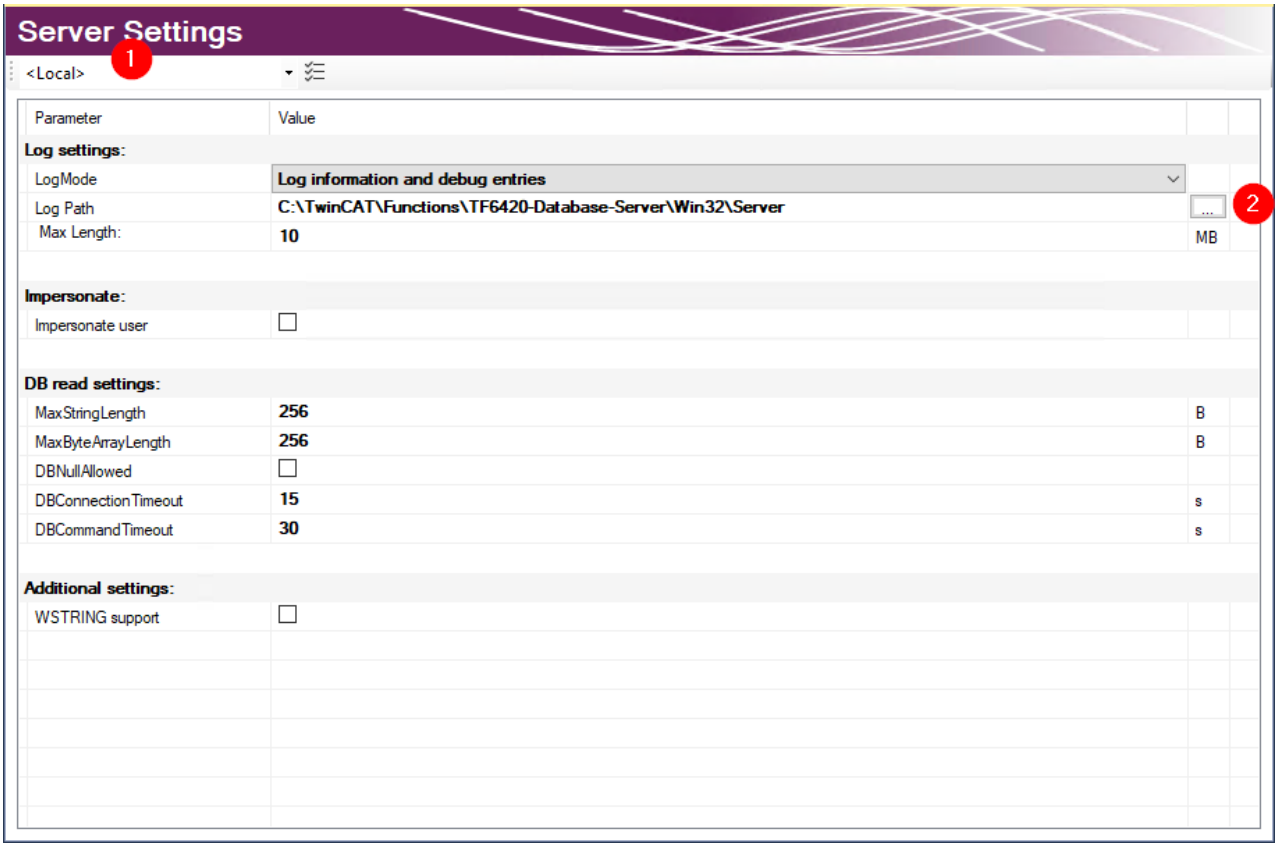
Die Toolbar besteht aus folgenden Elementen:



Toolstrip Button	Beschreibung
	Aktivieren der Konfiguration
	Konfiguration vom Zielgerät lesen
	Konfiguration in eine XML-Datei abspeichern
	Konfiguration aus einer XML-Datei einlesen
	Neue Datenbankkonfiguration hinzufügen
	Neue AutoLog-Gruppe hinzufügen
	Eventanzeige
	Datenbankpool
	AutoLog Viewer
	Information LogView
	SQL Query Editor

5.1.2.1.2 Projekteigenschaften

Editor für Server-Einstellungen



Im Editor **Server Settings** können Sie die Einstellungen für den TwinCAT Database Server bearbeiten. Diese Einstellungen betreffen den entsprechenden Server im Allgemeinen. Im Drop-down-Menü (1) können Sie das Zielsystem über die Ams-NetId angegeben. Dafür müssen Sie über TwinCAT eine Route zum Zielsystem anlegen. Wird eine fertige Konfiguration übertragen, werden die Einstellungen auf den TwinCAT Database Server dieses Zielsystems gespeichert.

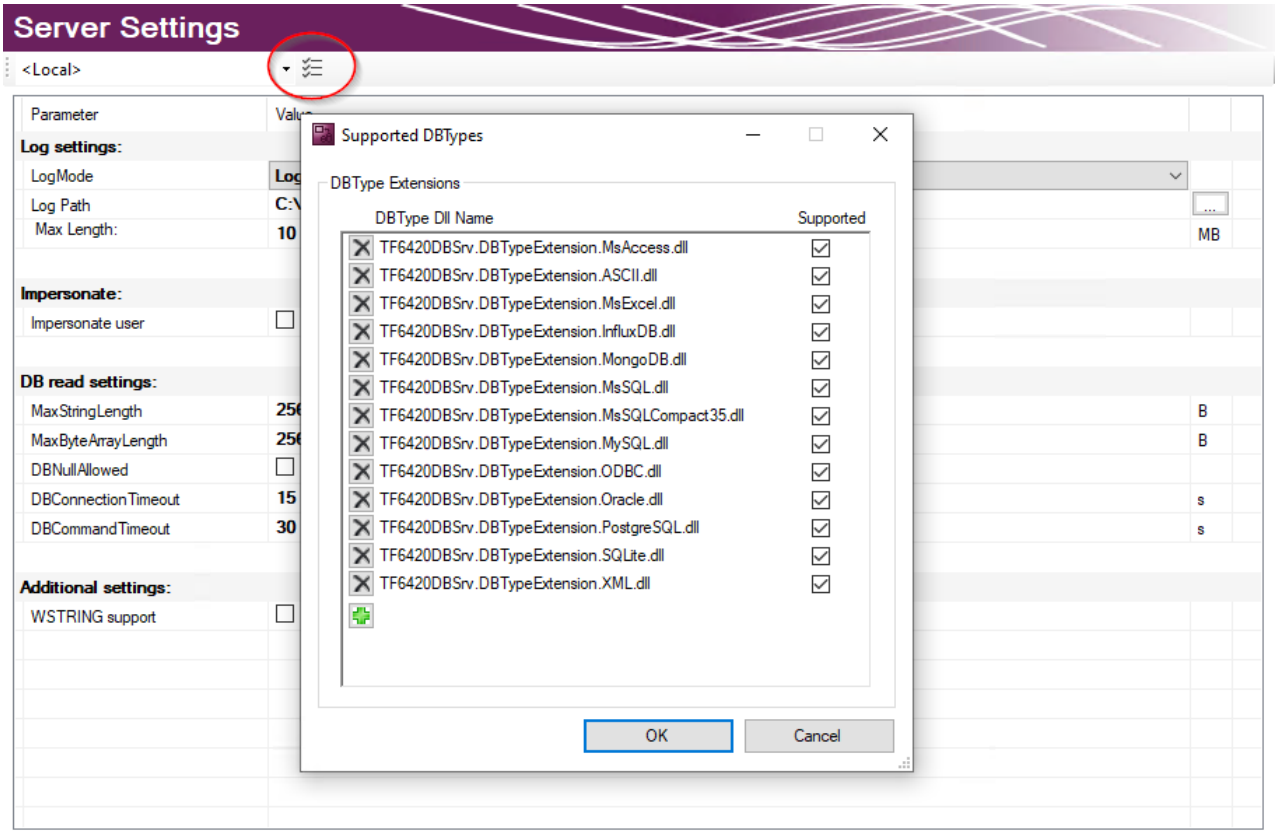
In den **Log settings** konfigurieren Sie Einstellungen zum Aufzeichnen von Fehlerfällen. In einem Fehlerfall erzeugt der Database Server einen detaillierten Eintrag in einer Logdatei. Diese können Sie mit dem Information Log Viewer [▶ 51] auslesen. In den **Log Settings** geben Sie einen Pfad zum Ablageort und die maximale Größe der Datei an. Zusätzlich können Sie die Genauigkeit des Logs beeinflussen. Wir empfehlen, aus Performancegründen, das Loggen nach erfolgter Fehleranalyse wieder abzuschalten, wenn es nicht mehr benötigt wird.

Bei Netzwerkzugriff auf dateibasierende Datenbanken wie Access Datenbanken oder SQL Compact Datenbanken müssen Sie die Option **Impersonate** setzen, damit sich der TwinCAT Database Server mit dem Netzwerklaufwerk verbinden kann. **Diese Funktion wird zurzeit nicht unter Windows CE unterstützt.**

Sie können weitere Einstellungen konfigurieren, um das Lesen aus der Datenbank zu steuern. Diese Einstellungen beziehen sich auf den TwinCAT Database Server auf dem Zielsystem:

MaxStringLength	Maximale String-Länge der Variablen in der SPS
MaxByteArrayLength	Maximale Byte-Array-Länge der Variablen in der SPS
DBNullAllowed	Gibt an, ob NULL-Werte im TwinCAT Database Server angenommen werden.
DBConnectionTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem Verbindungsaufbau von einem Verbindungsfehler ausgeht.
DBCommandTimeout	Gibt die Zeit an, wann der TwinCAT Database Server bei einem abgesendeten Kommando von einem Verbindungsfehler ausgeht. Bei hohen Datenmengen kann, je nach Datenbank und Infrastruktur ein Befehl durchaus sehr viel Zeit in Anspruch nehmen.

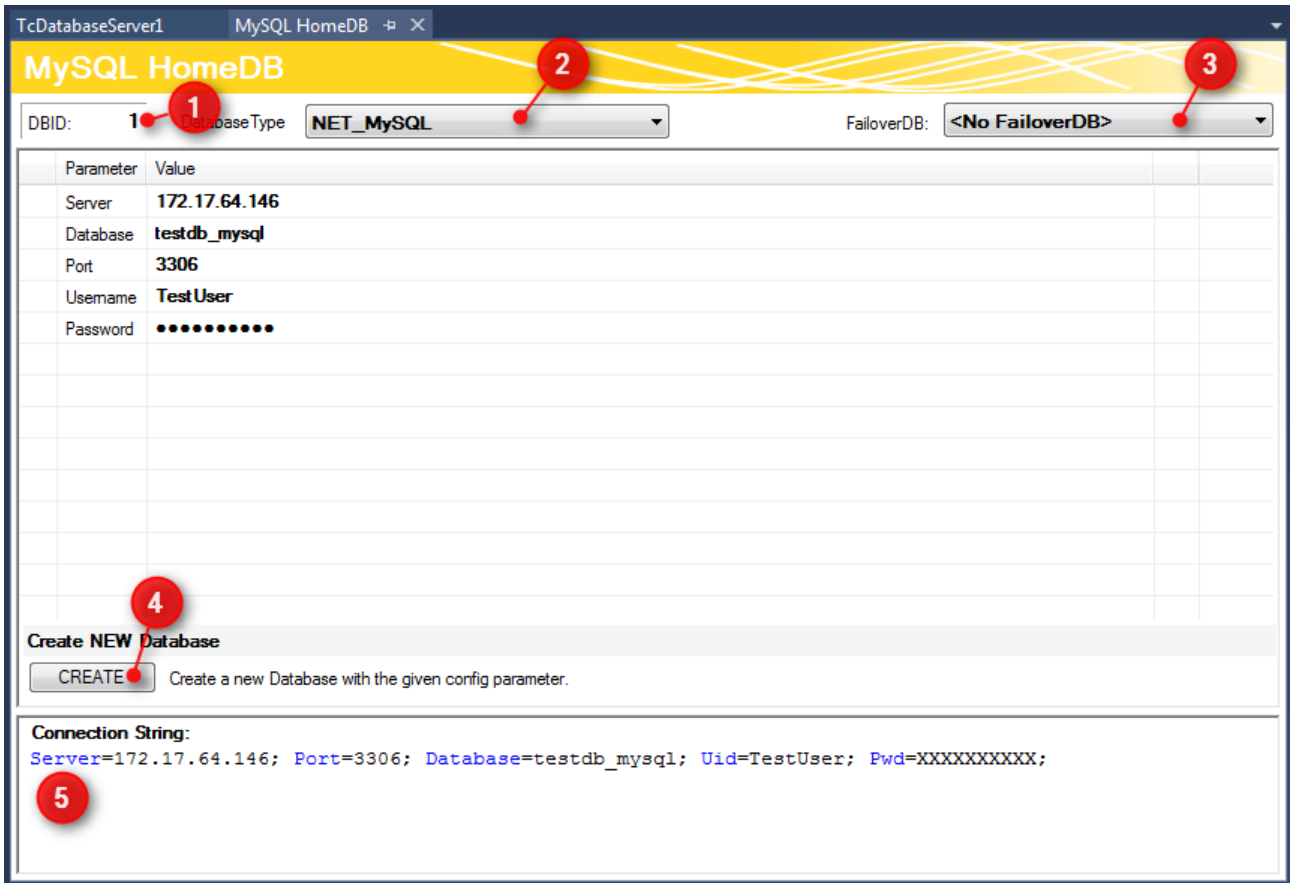
Unterstützte Datenbanktypen



Die installierten Datenbanktypen können in den Server Einstellungen ausgewählt werden. Standardmäßig sind alle installierten Datenbanken angewählt. Der TwinCAT 3 Database Server wird die entsprechenden Datenbankschnittstellen laden. Auf diese Weise können nicht genutzte Datenbanken auf dem Zielsystem abgewählt werden.

5.1.2.1.3 Datenbanken konfigurieren

Editor für Datenbankkonfigurationen

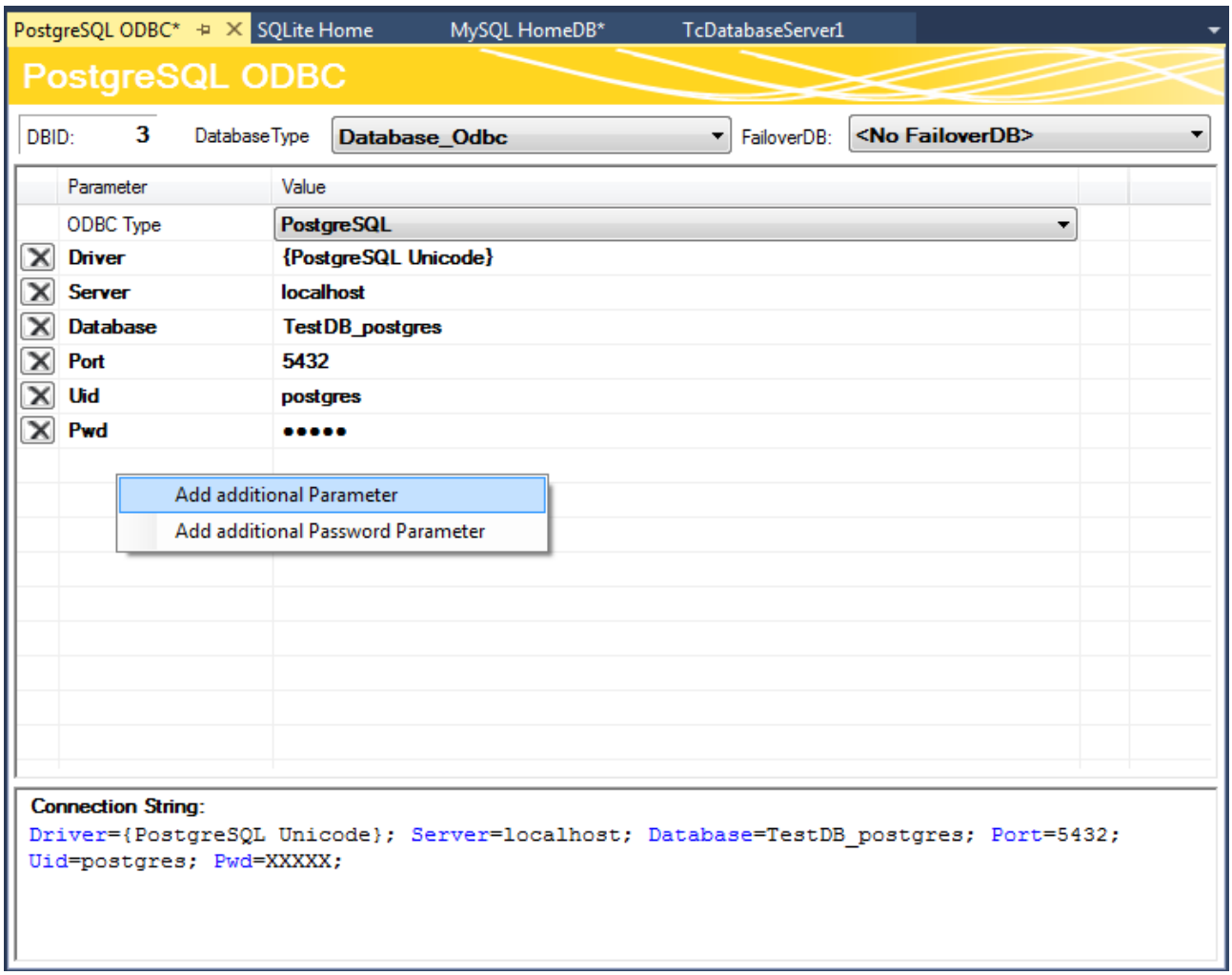


Im oberen Teil des Editors wird die Datenbank ID angezeigt (1), die für einige Funktionsbausteine in der SPS benötigt wird. Den Datenbanktyp der Zieldatenbank können Sie aus dem Drop-down-Menü auswählen (2). Hier können Sie auch die Odbc-Schnittstelle für eine Datenbank nutzen, die noch nicht unterstützt wird. Beachten Sie, dass je nach Datenbank nicht alle Funktionen des TwinCAT Database Servers gewährleistet werden.

Außerdem können Sie optional eine sogenannte FailOver-Datenbank (3) auswählen, welche im ‚Configure'-Mode beim Fehlerfall einspringt. Bei einem Verbindungsabbruch zum Netzwerk kann in einem solchen Fall automatisch gewährleistet werden, dass keine Daten verloren gehen und an anderer Stelle gespeichert werden.

Für jede Datenbank [▶ 126] stehen zusätzlich weitere einstellbare Parameter zur Verfügung. Je nach Datenbank wird ein Connection String (5) erstellt, welcher die Verbindung zur Datenbank beschreibt. Diese Anzeige dient zur Transparenz Ihrer eingestellten Parameter.

Über die Schaltfläche **CREATE** (4) können Sie eine neue Datenbank erstellen. Diese Funktion wird nur angezeigt, wenn sie von der jeweiligen Datenbank unterstützt wird.

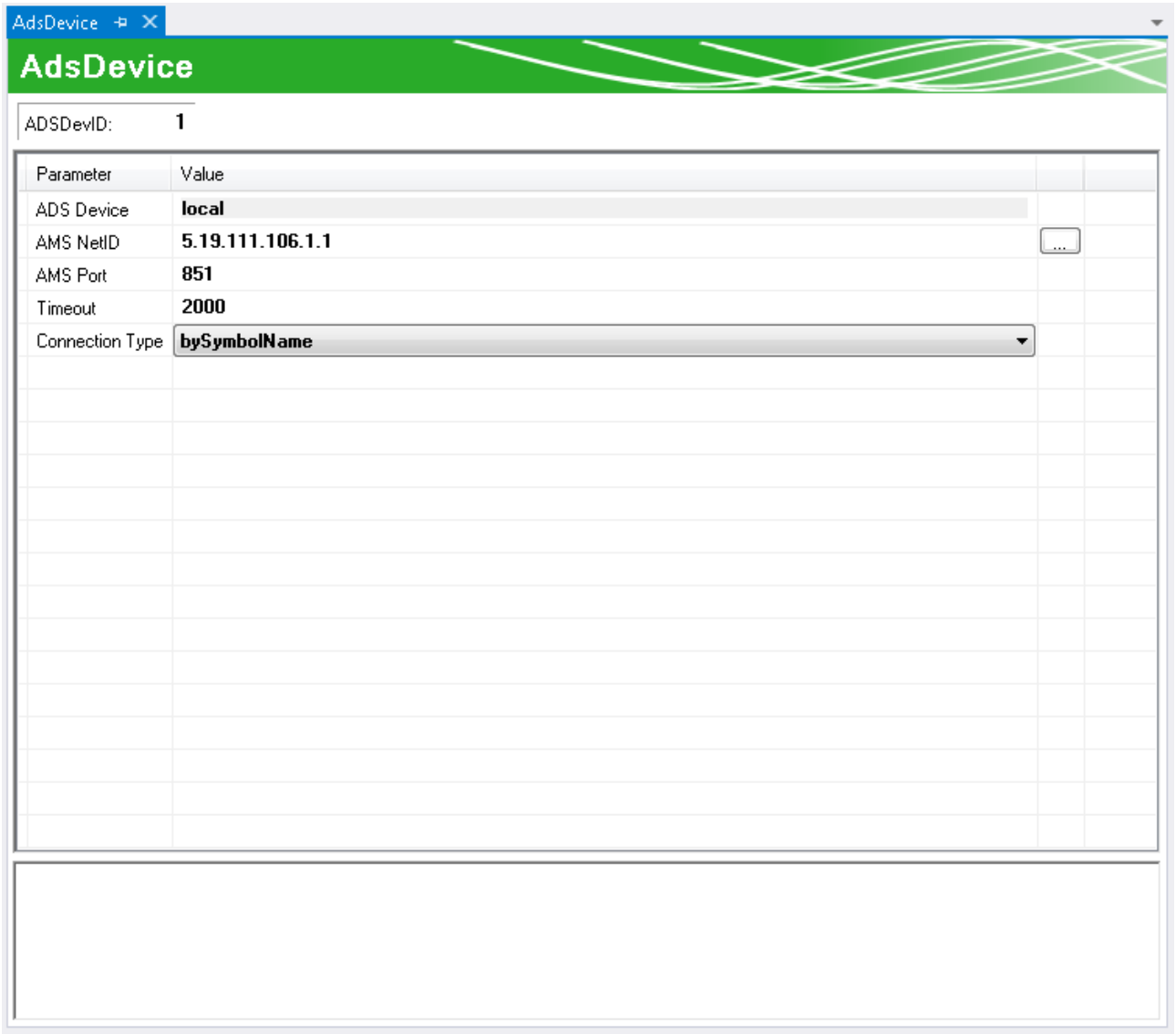


Sie können auch unbekannte Datenbanken mit einer ODBC-Schnittstelle konfigurieren. Dafür wählen Sie im in der Drop-down-Liste **ODBC Type** den Eintrag „Unknown Database“ und fügen über die Befehle im Kontextmenü Parameter hinzu. Diese können auch Passwörter beinhalten, welche dann verschlüsselt abgespeichert werden. Daraus kann der gewünschte Connection String zusammengestellt werden. Beachten Sie, dass nur begrenzte Funktionen des TwinCAT Database Servers genutzt werden können. Nur die expliziten Funktionsbausteine des SQL Expert Modes werden unterstützt.

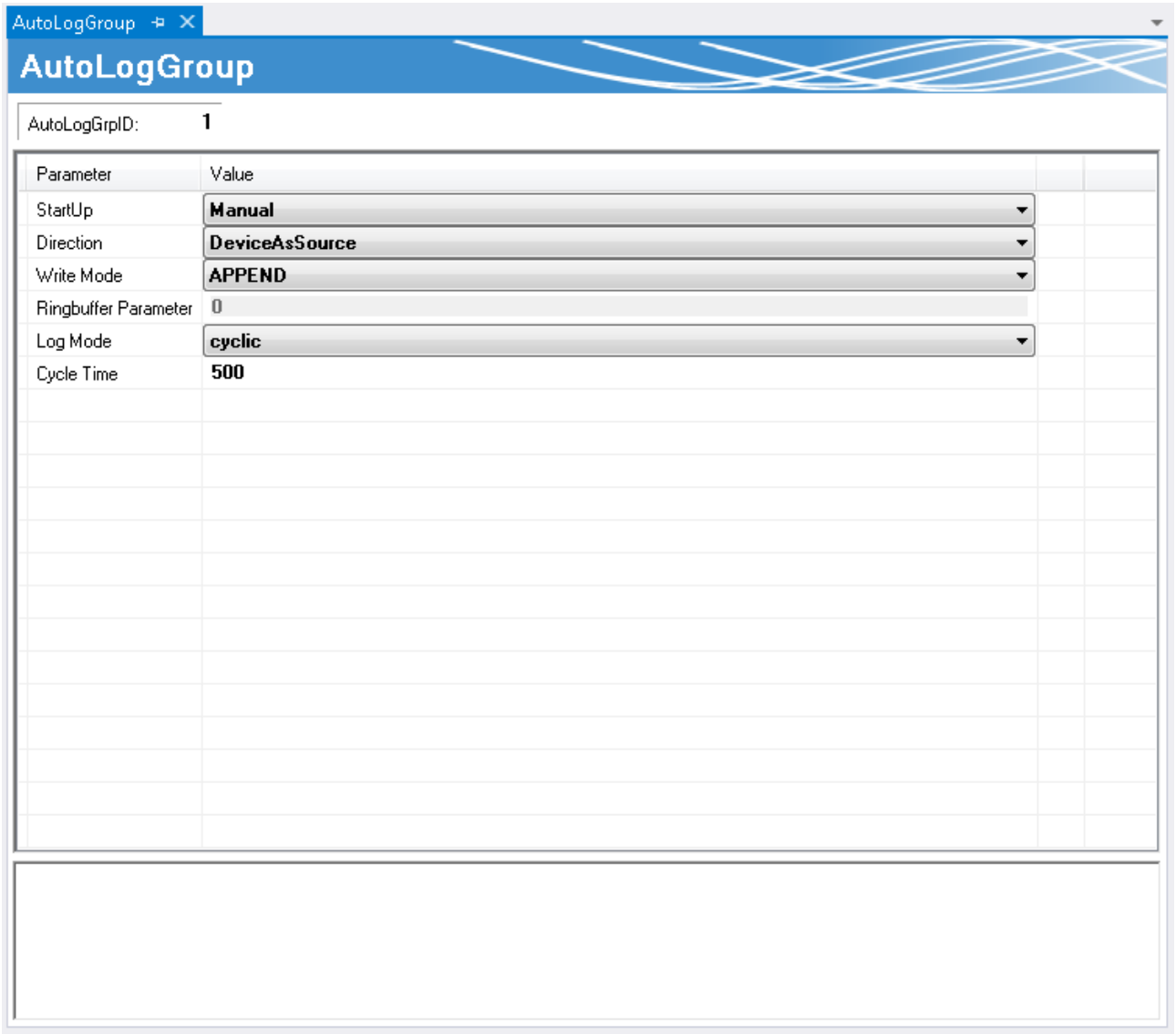
5.1.2.1.4 AutoLog-Gruppen konfigurieren

Ads Device konfigurieren

Das Ads Device wird automatisch unter eine AutoLog-Gruppe angelegt. Das Ads Device ist im häufigsten genutzten Anwendungsfall die SPS-Laufzeit. Folgende Einstellungen können im Editor getroffen werden:



ADS Device	Bezeichnung des ADS-Zielgeräts.
AMS NetID	Adresse des Zielgeräts im TwinCAT-Netzwerk.
AMS Port	Port des Zielgeräts im TwinCAT-Netzwerk.
Timeout	Zeit, nach der von einem Verbindungsabbruch zum Zielgerät ausgegangen wird.
Connection Type	bySymbolName: Verbindung wird anhand des Namens des Symbols hergestellt. byIndexGroup: Verbindung wird anhand des Speicherindex hergestellt.

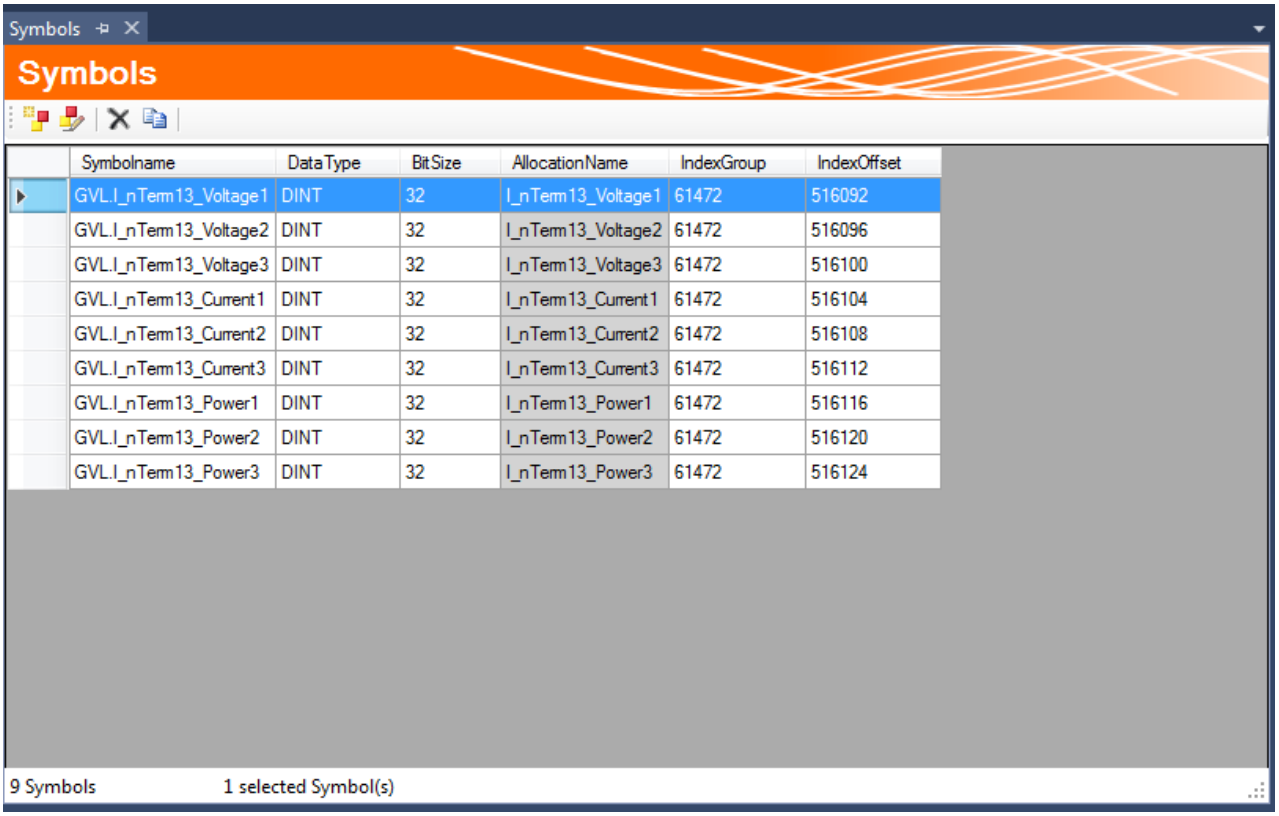


StartUp	Der AutoLog-Modus kann über den manuellen Weg (durch einen Befehl in der SPS oder aus dem Konfigurator) oder automatisch beim Systemstart eingeschaltet werden.
Direction	Das eingestellte ADS-Gerät dient als Datenziel oder als Datenquelle.
Write Mode	Die Daten können in einer Datenbank zeilenweise angehängt, auf zeitlicher Basis oder nach Anzahl in einem Ringpuffer gehalten oder einfach an der entsprechenden Position aktualisiert werden.
Ringbuffer Parameter	Je nach Einstellung stellt dieser Parameter die Zeit oder die Zyklen dar, nach der der Ringbuffer aktualisiert wird.
Log Mode	Entweder wird die Variable nach Ablauf eines gewissen Zyklus oder auf Änderung geschrieben.
Cycle Time	Zykluszeit, nach welcher die Variable geschrieben wird.

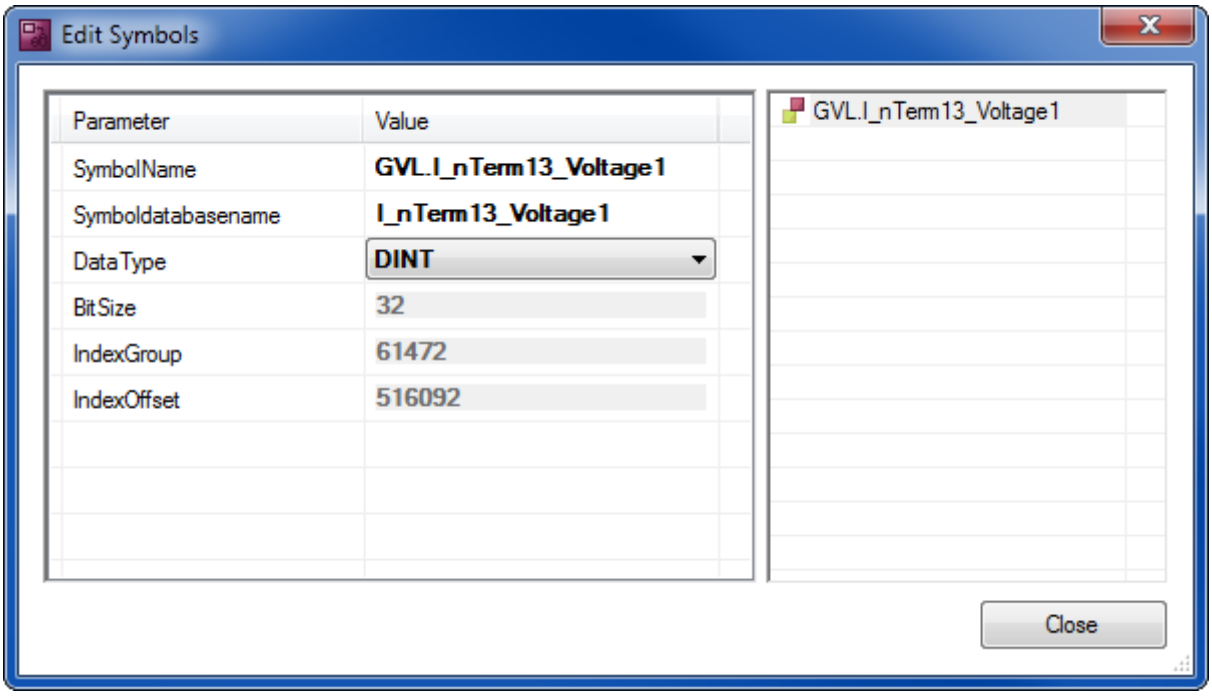
Eine neue AutoLog-Gruppe für die Datenbankkonfiguration können Sie mithilfe des Kommandos **Add new AutologGroup** im Kontextmenü einer Datenbankkonfiguration oder über die Toolbar hinzufügen. Diese AutoLog-Gruppen beziehen sich auf die übergeordnete Datenbank.

Symbole konfigurieren

Je nachdem, ob das ADS-Gerät Datenziel oder Datenquelle ist, werden die Symbole, die Sie hier einstellen, in die Datenbank geschrieben oder aus der Datenbank ausgelesen. Für einen komfortablen Zugriff können Sie den TwinCAT Target Browser [▶ 52] verwenden. Hier können die Symbole auf dem Target gesucht und per drag-and-drop zwischen den beiden Tools kommuniziert werden.



Sie könne Symbole auch manuell zur Symbolgruppe hinzufügen oder bearbeiten. Je nachdem, ob im ADS-Gerät der Verbindungstyp über die Symbolnamen oder die Index-Gruppen ausgewählt wurden, werden entsprechende Informationen benötigt. Dabei wird immer vom ADS-Gerät ausgegangen.



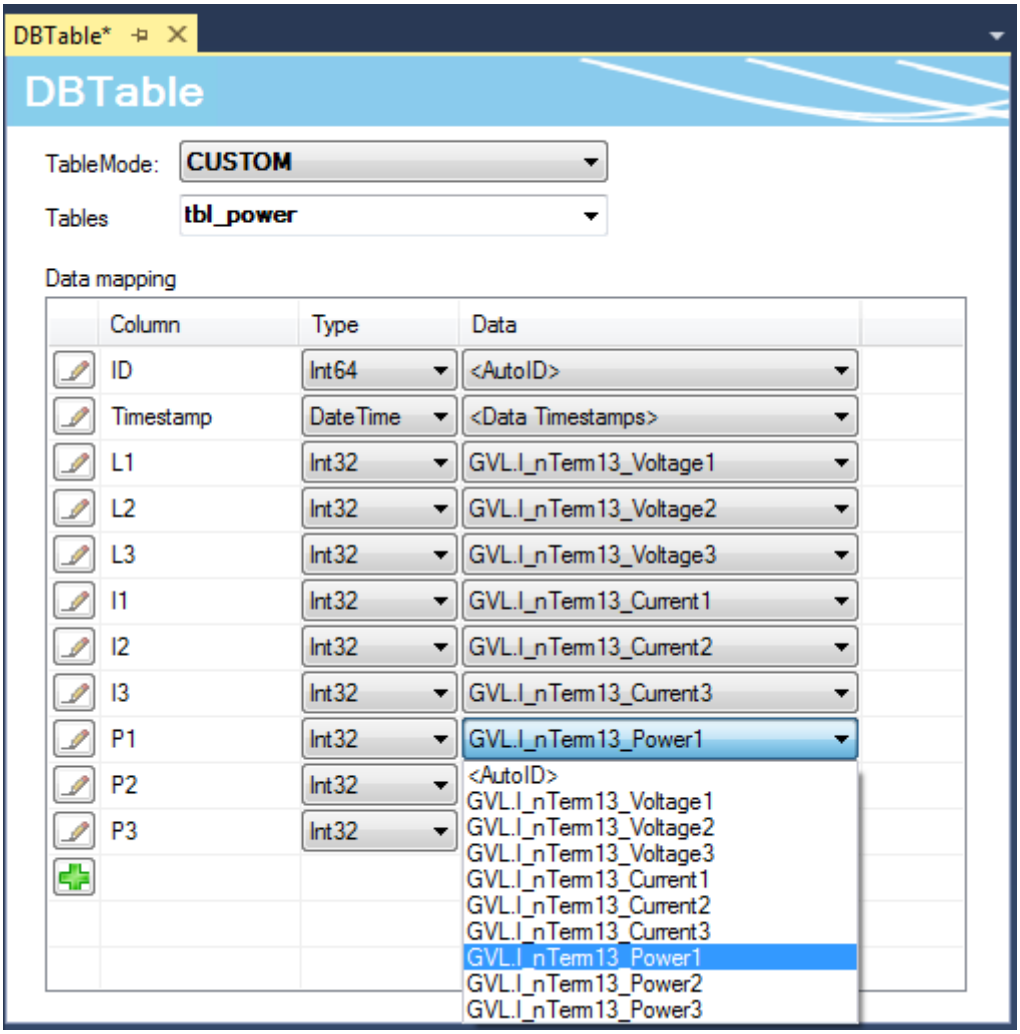
SymbolName	ausgehend vom einstellten ADS-Gerät wird das Symbol angesprochen
Symboldatabasename	Name der Variable in der Datenbanktabelle
DataType	SPS-Datentyp des Symbols
BitSize	Bit-Größe des Symbols (wird bei automatisch für die Datentypen eingestellt)
IndexGroup	Indexgruppe im TwinCAT-System
IndexOffset	Indexoffset im TwinCAT-System

Tabelle konfigurieren

Die Tabelle in einer Datenbank kann nach einer Standardtabellenstruktur oder nach einer individuellen Struktur aufgebaut sein.

Die entsprechende Tabelle können Sie aus einer Liste möglicher Tabellen auswählen. Ist die Tabelle noch nicht vorhanden, können Sie diese mithilfe des SQL Query Editors erzeugen. Falls Sie die Standardtabellenstruktur auswählen, zeigt Ihnen ein blauer Haken an, ob die ausgewählte Tabelle dieser Struktur entspricht.

Der spezifische Tabellentyp bietet Ihnen die Möglichkeit, die einzelnen Symbole, welche in der Symbolgruppe eingestellt wurden, auf die Tabellenspalten in der Datenbank beliebig zu verteilen. Wird ein Datensatz nun während des AutoLog-Modus in die Datenbank geschrieben, werden die aktuellen Werte der Symbolgruppe zum Abtastzeitpunkt in der entsprechenden Spalte der Tabelle gespeichert.



5.1.2.1.4.1 Schreibrichtungsmodus

Der TwinCAT Database Server verfügt über vier verschiedene Schreibrichtungsmodi. Diese werden nachfolgend erläutert.

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 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 mithilfe 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

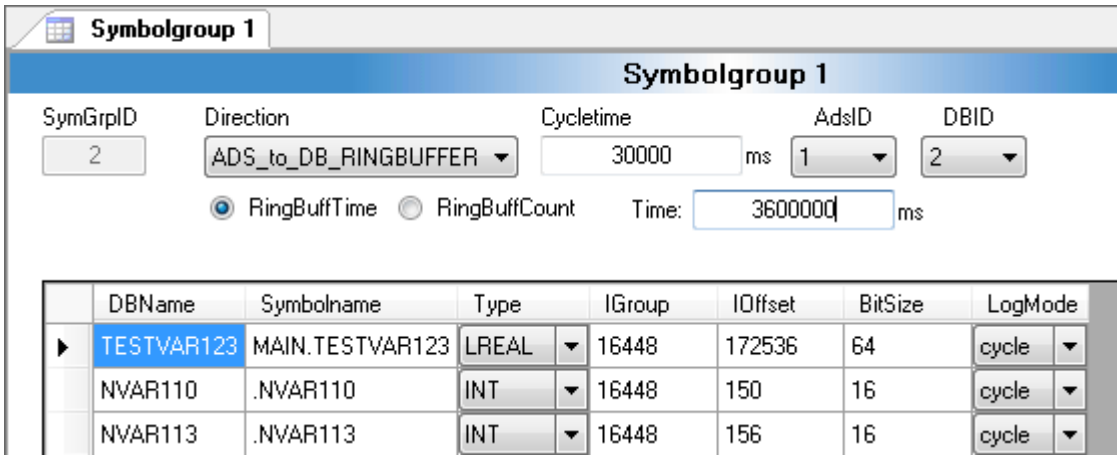
Bei diesem Modus kann eine Zeit angegeben werden, die das maximale Alter eines Datensatzes festlegt. Wird dieses Alter überschritten, wird der betroffene Datensatz gelöscht.

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.

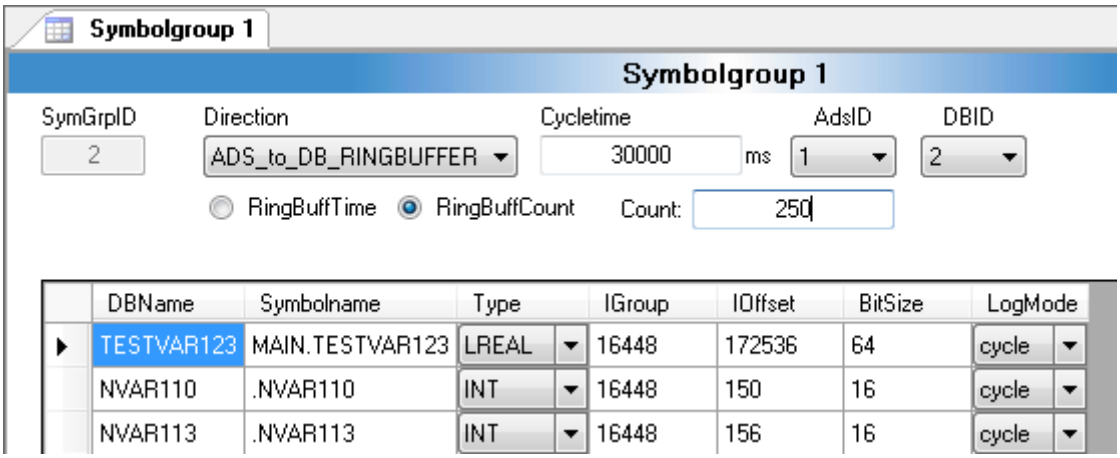
Deklarieren des RingBuffer Modus im XML-Konfigurationsdatei-Editor

RingBuffer_Time:



Die Zeit wird in Millisekunden angegeben.

RingBuffer_Count:



Deklarieren des RingBuffer-Modus im FB_DBWrite:

RingBuffer_Time:

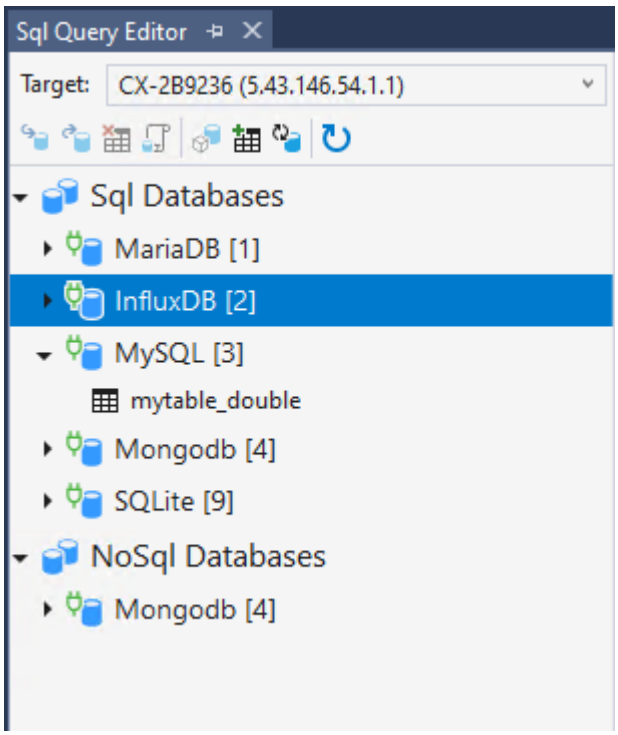
```
fbDBWrite(
    sNetID:= '',
    hDBID:= 2,
    hAdsID:= 1,
    sVarName:= 'MAIN.TESTVAR123',
    nIGroup:= ,
    nIOffset:= ,
    nVarSize:= ,
    sVarType:= ,
    sDBVarName:= 'TESTVAR123',
    eDBWriteMode:= eDBWriteMode_RingBuffer_Time,
    tRingBufferTime:= 3600000,
    nRingBufferCount:= ,
    bExecute:= TRUE,
    tTimeout:= T#15S,
    bBusy=> bBusy,
    bError=> bErr,
    nErrID=> nErrID,
    sSQLState=> stSQLState);
```

RingBuffer_Count:

```
fbDBWrite(
    sNetID:= '',
    hDBID:= 2,
    hAdsID:= 1,
    sVarName:= 'MAIN.TESTVAR123',
    nIGroup:= ,
    nIOffset:= ,
    nVarSize:= ,
    sVarType:= ,
    sDBVarName:= 'TESTVAR123',
    eDBWriteMode:= eDBWriteMode_RingBuffer_Count,
    tRingBufferTime:= ,
    nRingBufferCount:= 250,
    bExecute:= TRUE,
    tTimeout:= T#15S,
    bBusy=> bBusy,
    bError=> bErr,
    nErrID=> nErrID,
    sSQLState=> stSQLState);
```

5.1.2.1.5 SQL Query Editor

Der SQL Query Editor ist ein Tool des Database Servers, um die Entwicklung Ihrer Applikation zu unterstützen. Mit dem Tool können Verbindungen und SQL-Befehle getestet und die Kompatibilität zwischen SPS und Datenbanken geprüft werden.



Nachdem der TwinCAT Database Server des Zielsystems gewählt wird, lädt der SQL Query Editor die aktuelle Datenbankkonfiguration und die Tabellen der erfolgreich verbundenen Datenbanken. Je nachdem, ob die Datenbank die SQL und die NoSQL-Schnittstelle (vom TwinCAT Database Server) unterstützt, wird sie unter der jeweiligen Kategorie aufgeführt.

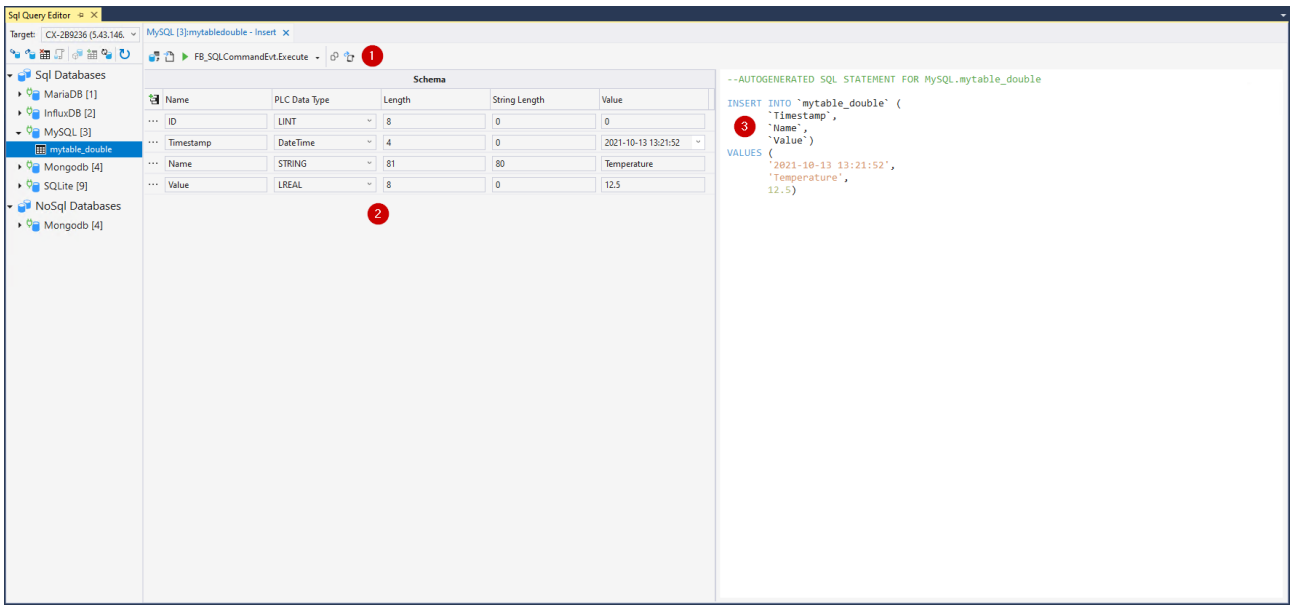
Unter der Auswahl des Zielsystems befindet sich eine Statusleiste mit den verfügbaren Befehlen:

Tabellenebene	
Insert Arbeitsbereich	Öffnet den Insert-Arbeitsbereich, um Datensätze mit SQL in die ausgewählte Tabelle zu schreiben.
Select Arbeitsbereich	Öffnet den Select-Arbeitsbereich, um Datensätze mit SQL aus der gewählten Tabelle zu lesen.
Delete/Drop Arbeitsbereich	Öffnet den Delete/Drop-Arbeitsbereich, um Datensätze mit SQL aus der gewählten Tabelle zu löschen oder ganze Tabellen zu löschen.
NoSQL Arbeitsbereich	Öffnet den NoSQL-Arbeitsbereich, um NoSQL-spezifische Abfragen auszuführen.
Datenbankebene	
Stored Procedure Arbeitsbereich	Öffnet den Stored Procedure-Arbeitsbereich, um gespeicherte Prozeduren der Datenbank auszuführen.
Tabellen Arbeitsbereich	Öffnet den Tabellen Arbeitsbereich, um neue Tabellen in der ausgewählten Datenbank zu erstellen.
Tabellen aktualisieren	Aktualisiert die verfügbaren Tabellen der ausgewählten Datenbank.
Allgemein	
Datenbanken aktualisieren	Aktualisiert den gesamten Datenbankbaum.

Die Arbeitsbereiche werden rechts neben dem Baum unter dem entsprechenden Reiter geöffnet. Auch von der gleichen Tabelle können mehrere Reiter zu einem Zeitpunkt geöffnet werden.

Insert-Arbeitsbereich

Der Insert-Arbeitsbereich ermöglicht das Schreiben von Daten in die ausgewählte Tabelle über die TwinCAT Database Server Schnittstelle für SQL-Funktionsbausteine.



Im unteren Bereich (2) befindet sich eine Tabelle mit den einzelnen Datensymbolen im zu schreibenden Datensatz. Der Name, PLC-Datentyp, die Bytelänge sowie der Wert können hier bestimmt werden. Die eingetragenen Werte werden anschließend über den Befehl zum Generieren der SQL-Anweisung verwenden.

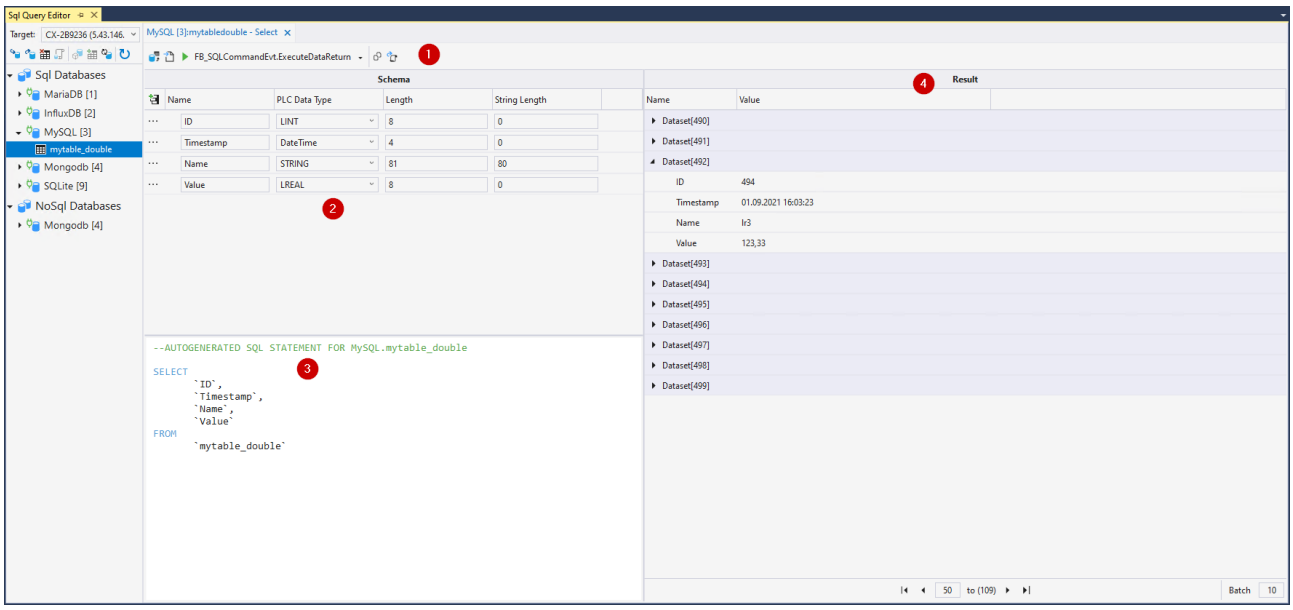
Diese SQL-Anweisung wird dann in einem Textfeld (3) zur Verfügung gestellt. Je nach Syntax der Datenbank kann der Inhalt unterschiedlich ausfallen.

In der oberen Statusleiste befinden sich die Kommandos zum Interagieren mit dem TwinCAT Database Server (1).

Kommando	Beschreibung
Lese Tabellen Schema	Liest das Tabellenschema der Tabelle des Arbeitsbereichs aus.
Generiere SQL-Anweisung	Generiert abhängig von der Datenbanksyntax aus der vorliegenden Tabelle die SQL-Anweisung.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die im Textfeld (3) stehende Anweisung aus.
Kopieren der Anweisung	Kopiert die im Textfeld (3) stehende Anweisung als TwinCAT kompatible Syntax.
Exportieren als Struktur	Exportiert die Struktur der Tabelle der Eingabewerte zu einem TwinCAT 3 kompatiblen DUT.

Select-Arbeitsbereich

Der Select-Arbeitsbereich ermöglicht das Lesen von Daten in die ausgewählte Tabelle über die TwinCAT Database Server Schnittstelle für SQL-Funktionsbausteine.



Im unteren Bereich (2) befindet sich eine Tabelle mit den einzelnen Datensymbolen im zu lesenden Datensatz. Der Name, PLC-Datentyp, sowie die Bytelänge können hier bestimmt werden. Diese Informationen werden anschließend zum Interpretieren der Daten benötigt.

Diese SQL-Anweisung wird dann in einem Textfeld (3) zur Verfügung gestellt. Je nach Syntax der Datenbank kann der Inhalt unterschiedlich ausfallen.

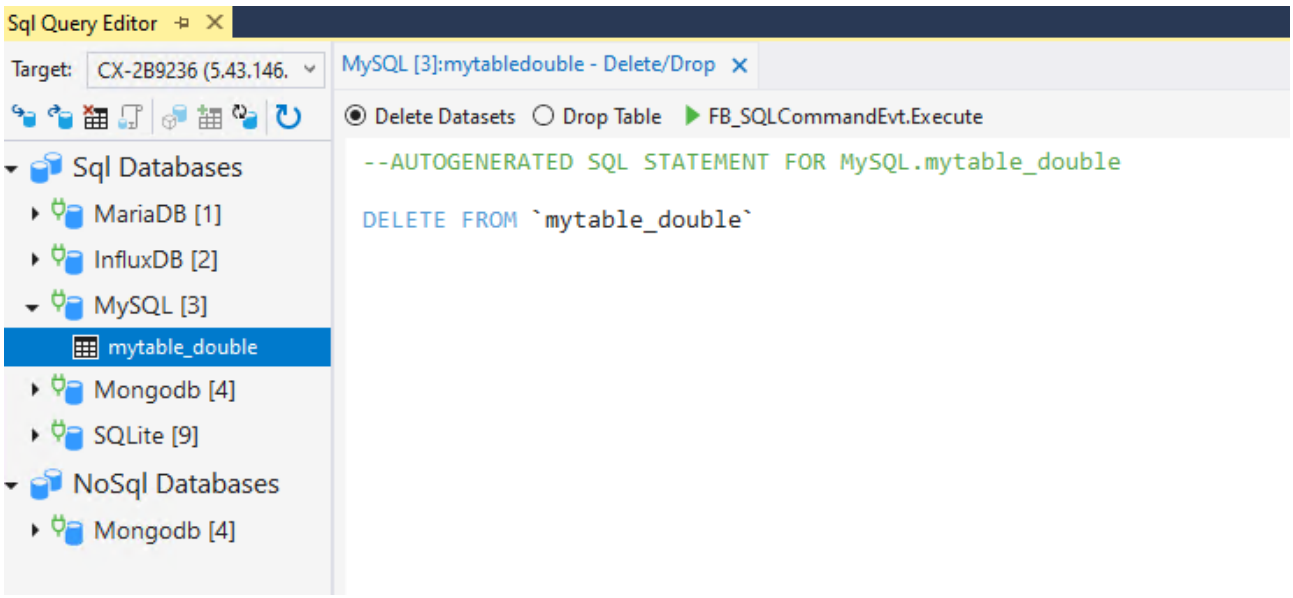
Im Ergebnis-Feld (4) werden nach Ausführung der Anweisung die Daten angezeigt. Falls mehrere Ergebnisse zurückgeliefert werden, können diese unter über die Seiten durchgeschaltet werden.

In der oberen Statusleiste befinden sich die Kommandos zum Interagieren mit dem TwinCAT Database Server (1).

Kommando	Beschreibung
Lese Tabellen Schema	Liest das Tabellenschema der Tabelle des Arbeitsbereichs aus.
Generiere SQL-Anweisung	Generiert abhängig von der Datenbanksyntax aus der vorliegenden Tabelle die SQL-Anweisung.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die im Textfeld (3) stehende Anweisung aus.
Kopieren der Anweisung	Kopiert die im Textfeld (3) stehende Anweisung als TwinCAT kompatible Syntax.
Exportieren als Struktur	Exportiert die Struktur der Tabelle der Eingabewerte zu einem TwinCAT 3 kompatiblen DUT.

Delete/Drop-Arbeitsbereich

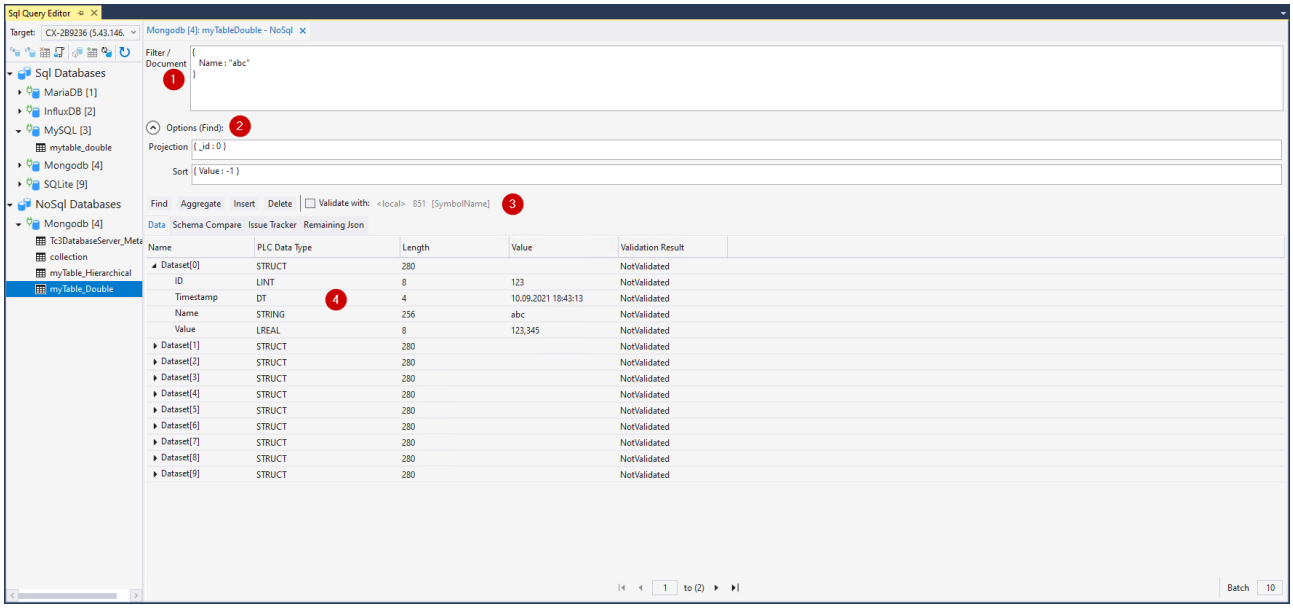
Der Delete/Drop-Arbeitsbereich bietet die Option SQL-Anweisungen abzusetzen, um entweder Daten aus einer Tabelle oder die gesamte Tabelle aus der Datenbank zu löschen.



Hierfür kann zwischen den beiden Optionen in der Statusleiste ausgewählt werden. Die der Datenbank entsprechenden Syntax wird daraufhin im Anweisungsfeld generiert. Um diese mit der TwinCAT Database Server Schnittstelle auszuführen, steht der Schalter **FB_SQLCommandEvt.Execute** zur Verfügung.

NoSql-Arbeitsbereich

Der NoSql-Arbeitsbereich unterstützt die speziellen Funktionen von NoSql-Datenbanken bzw. der TwinCAT Database Server NoSQL-Schnittstelle.



ID	Bezeichnung	Funktion
1	Filter/Document	Je nachdem welche Funktion genutzt wird, agiert dieses Eingabefeld als Dokument oder als Filter im Json-Format. Falls Sie einen Find ausführen möchten und zusätzlich eine Projektion oder eine Sortierung vornehmen möchten, lassen sich mit den darunterliegenden Options(Find) diese Felder füllen.
2	Options (Find)	Beschreibt zusätzliche Parameter für die Find-Funktion, wie die Projektion oder Sortierung.
3	Steuerelemente	Steuerelemente zur Interaktion mit der TwinCAT Database Server Schnittstelle für NoSQL.
4	Datenanzeige	Auflistung von zurückgelieferten Daten. Die Navigation ermöglicht die Iteration durch die verfügbaren Seiten.

Find: Führt eine Suchabfrage mit dem im Textfeld (1) eingetragenen Filter aus. Optional kann ebenfalls über die Options(Find) -Felder eine Projektion oder eine Sortierung vorgenommen werden. Hierbei werden Daten zurückgeliefert und in der Datenanzeige (4) aufgeführt. Die Syntax der Filter ist dabei datenbankspezifisch.

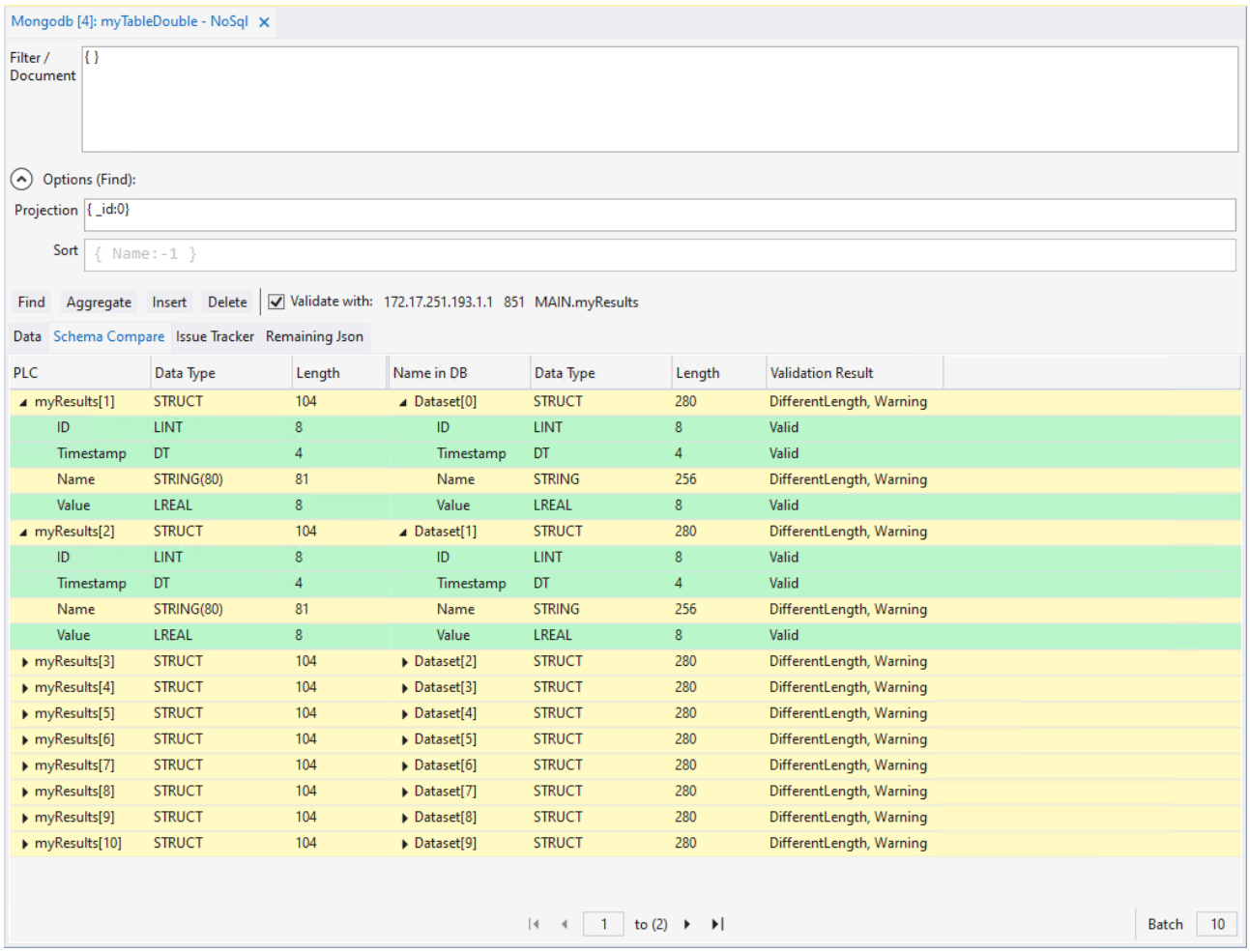
Aggregate: Führt eine Aggregation mit dem im Textfeld (1) eingetragenen Parametern aus. Hierbei werden Daten zurückgeliefert und in der Datenanzeige (4) aufgeführt. Die Syntax der Filter ist dabei datenbankspezifisch.

Insert: Führt eine Insert-Abfrage des im Textfeld (1) eingetragenen (Json-)Dokuments oder Dokument-Array aus. Diese werden dann in die Collection geschrieben.

Delete: Führt eine Delete-Abfrage auf die Daten aus, welche mit dem Filter im Textfeld (1) gefunden wurden. Die gefundenen Daten werden aus der Collection gelöscht.

Validate: Wird diese Option angewählt werden die Datenabfragen nicht automatisch nach ihrem eigenen Schema geparst, sondern versucht diese Daten auf die Struktur des Symbols aus der SPS abzubilden, welches über diese Parameter angegeben wurde.

Bei letzterer Funktion kann eine Find-Abfrage zu Konflikten führen. Im Gegensatz zu Strukturen im SPS-Prozessabbild, müssen Datensätze in NoSql-Datenbanken keinem festen Schema folgen. Möglicherweise besitzen abgefragte Dokumente keine Daten zu einem bestimmten Element in der SPS-Struktur. Oder der Datensatz trägt Daten, welche nicht in der SPS-Struktur vorhanden sind. Zugeordnet werden diese Daten über den Namen bzw. dem Attribut „ElementName“ in der SPS.



PLC	Data Type	Length	Name in DB	Data Type	Length	Validation Result
myResults[1]	STRUCT	104	Dataset[0]	STRUCT	280	DifferentLength, Warning
ID	LINT	8	ID	LINT	8	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
Name	STRING(80)	81	Name	STRING	256	DifferentLength, Warning
Value	LREAL	8	Value	LREAL	8	Valid
myResults[2]	STRUCT	104	Dataset[1]	STRUCT	280	DifferentLength, Warning
ID	LINT	8	ID	LINT	8	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
Name	STRING(80)	81	Name	STRING	256	DifferentLength, Warning
Value	LREAL	8	Value	LREAL	8	Valid
myResults[3]	STRUCT	104	Dataset[2]	STRUCT	280	DifferentLength, Warning
myResults[4]	STRUCT	104	Dataset[3]	STRUCT	280	DifferentLength, Warning
myResults[5]	STRUCT	104	Dataset[4]	STRUCT	280	DifferentLength, Warning
myResults[6]	STRUCT	104	Dataset[5]	STRUCT	280	DifferentLength, Warning
myResults[7]	STRUCT	104	Dataset[6]	STRUCT	280	DifferentLength, Warning
myResults[8]	STRUCT	104	Dataset[7]	STRUCT	280	DifferentLength, Warning
myResults[9]	STRUCT	104	Dataset[8]	STRUCT	280	DifferentLength, Warning
myResults[10]	STRUCT	104	Dataset[9]	STRUCT	280	DifferentLength, Warning

In der Registerkarte **Schema Compare** können die Unterschiede der Daten nachvollzogen werden. Im obigen Beispiel ist zu erkennen, dass im Falle des zurückgelieferten Dokuments in der PLC-Struktur die Variable „Name“ eine andere Datentyplänge hat als die der Datenbank. Die entsprechenden Farben zeigen die Gewichtung des Konflikts:

Rot: Es sind zu viele oder zu wenige Daten vorhanden.

Gelb: Die Bytelänge des Datensatzes stimmt nicht überein oder darunterliegende Datensätze sind übrig oder nicht vorhanden.

Grün: keine Konflikte

Diese Konflikte werden auch als Liste unter der Registerkarte **Issue Tracker** aufgeführt. Sie kann auch als String-Array bei Bedarf in die SPS eingelesen werden.

In der Registerkarte **Remaining Json** werden übrig gebliebene Datensätze als Json zurückgegeben. Auch diese Informationen können als String in die SPS gelesen werden.

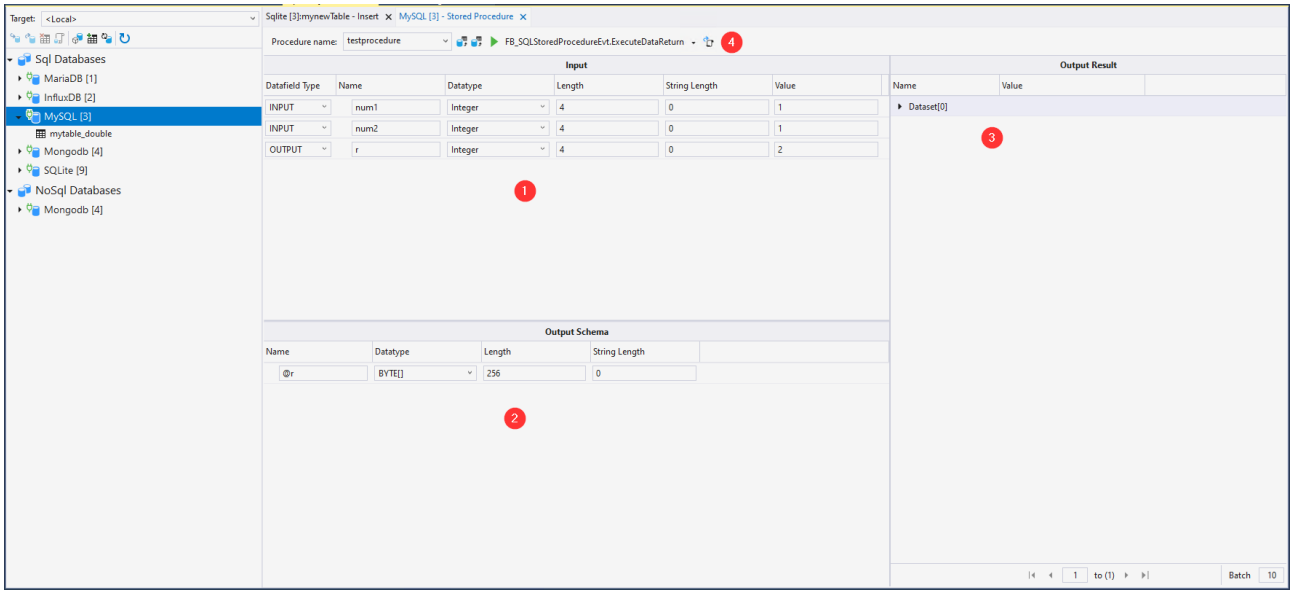
Über die Steuerelemente in der Statusleiste kann, wie von den anderen Datendarstellungen bekannt, durch die Daten iteriert werden. Dabei kann die Menge der gleichzeitig angezeigten Datensätze angegeben werden.

Stored Procedure-Arbeitsbereich

Der TwinCAT Database Server unterstützt „Stored Procedures“, die viele Datenbanken bereitstellen, um komplexere Abfragen auf der Datenbankebene zu verarbeiten oder eine vereinfachte Schnittstelle zur Verfügung zu stellen.

Falls **Stored Procedures** in der Datenbank vorhanden sind, werden diese in der Dropdown-Liste der Statusleiste (4) aufgeführt.

Darunter befindet sich die Tabelle für die Eingangsparameter (1), sowie für das Ausgabe-Schema (2). Zusätzlich gibt es eine Ansicht für die Ausgabeergebnisse (3). Bei erfolgreicher Ausführung der **Stored Procedure** werden die Ergebnisse hier angezeigt.

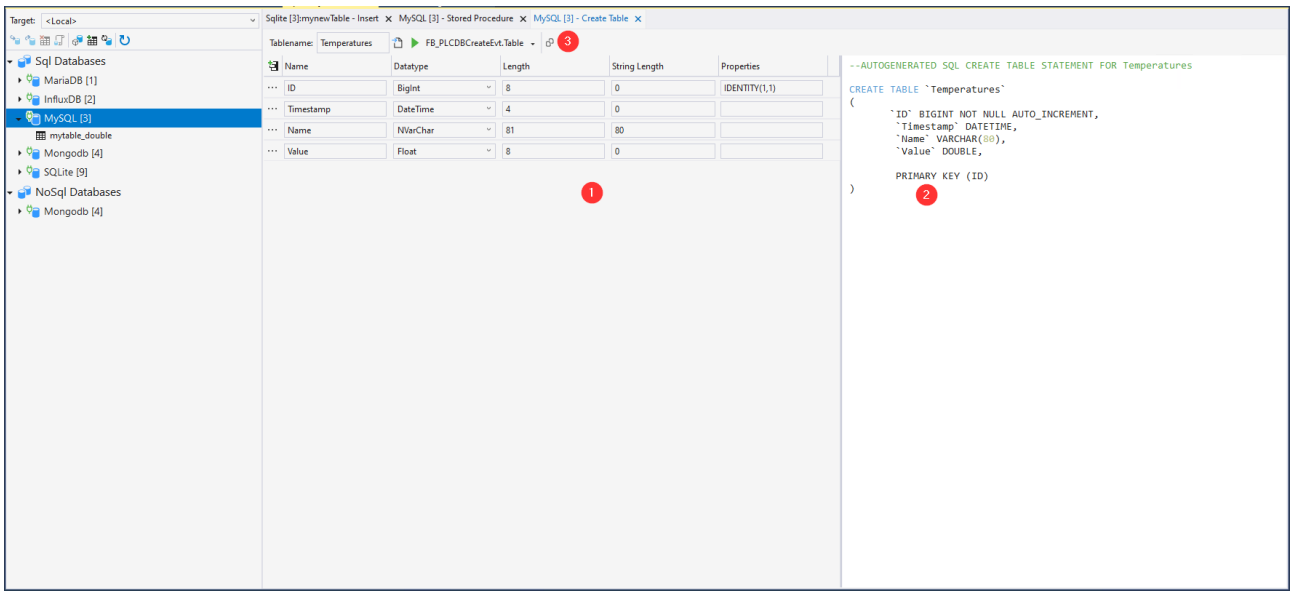


Die Statusleiste hat folgende Befehle:

Kommando	Beschreibung
Lese Stored Procedure Eingangsschema	Liest das Eingangsparameterschema aus. Die Ergebnisse werden in der Tabelle 1 aufgeführt.
Lese Stored Procedure Ausgangsschema	Liest das Ausgangsparameterschema aus. Die Ergebnisse werden in der Tabelle 2 aufgeführt. **Info:** Hierfür ist die Ausführung der Stored Procedure notwendig. Hierbei können je nach Programmierung Daten verändert werden.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die Stored Procedure aus.
Exportieren als Struktur	Exportiert die Struktur der Tabelle zu einem TwinCAT 3 kompatiblen DUT.

Table-Arbeitsbereich

Der Table-Arbeitsbereich dient zur Erstellung von neuen Tabellen.

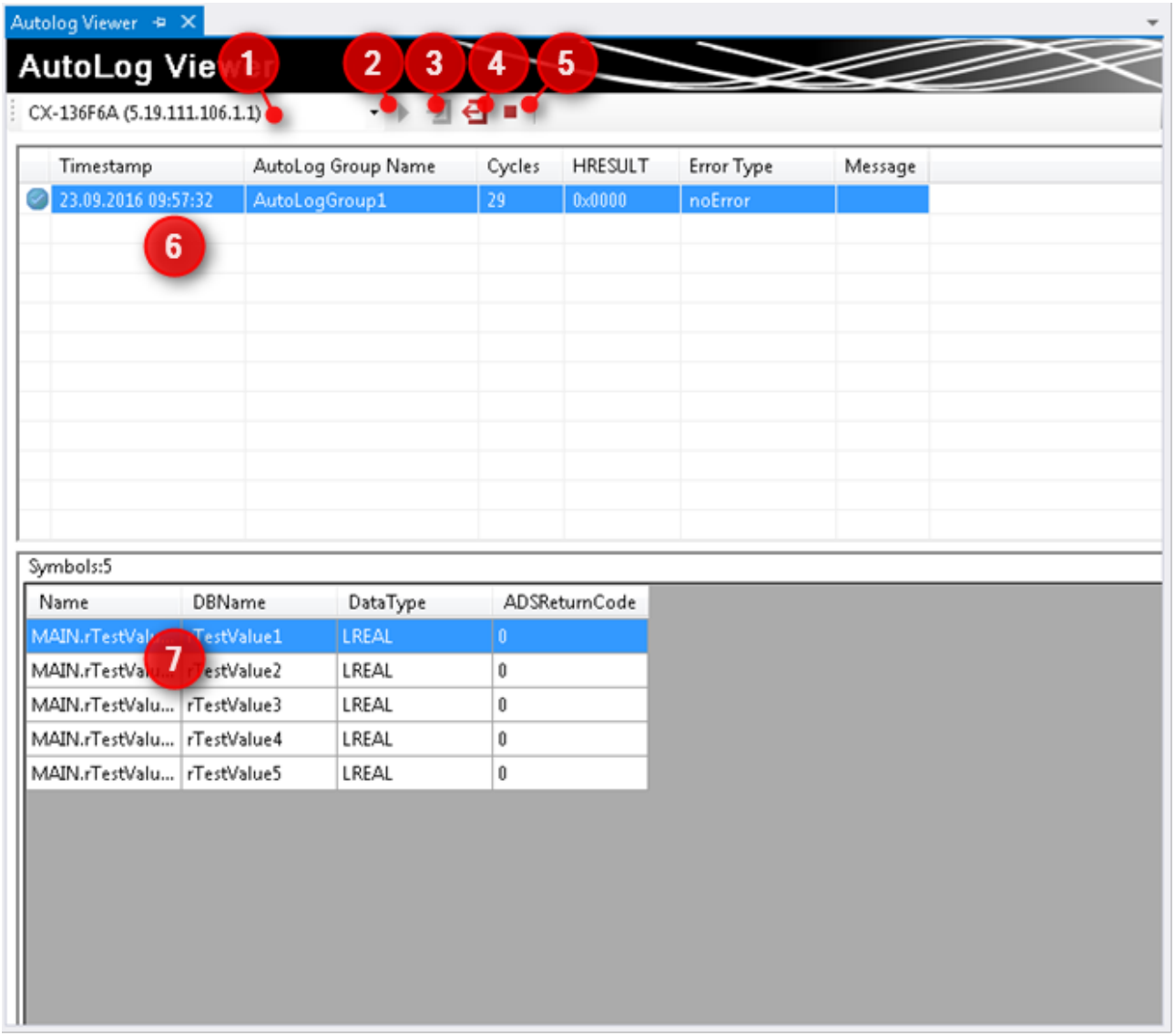


Hier kann die Tabellenstruktur (1) erstellt werden und daraus eine SQL-Anweisung im entsprechenden Feld (2) generiert werden. Hierfür kann die Statusleiste (3) mit folgenden Kommandos verwendet werden:

Kommando	Beschreibung
Tabellenname	Bestimmt den Tabellennamen der neuen Tabelle.
Generiere SQL-Anweisung	Generiert abhängig von der Datenbanksyntax aus der vorliegenden Tabelle die SQL-Anweisung.
Ausführung	Führt über die jeweilige Schnittstelle des TwinCAT Database Server die Stored Procedure aus.
Kopieren der Anweisung	Kopiert die im Textfeld (2) stehende Anweisung als TwinCAT kompatible Syntax.

5.1.2.1.6 AutoLog Live View

Der AutoLog Viewer des TwinCAT Database Server ist ein Tool, um den AutoLog-Modus zu steuern und zu überwachen. Ähnlich wie bei der TwinCAT SPS können Sie sich auf ein Zielsystem einloggen. Im eingeloggten Zustand kann der AutoLog-Modus gestartet oder gestoppt werden. Im unteren Bereich des Fensters werden Informationen über den aktuellen Zustands des Loggens mitgeteilt. Durch das Selektieren einer AutoLog-Gruppe werden weitere Informationen über die geloggten Symbole angezeigt.

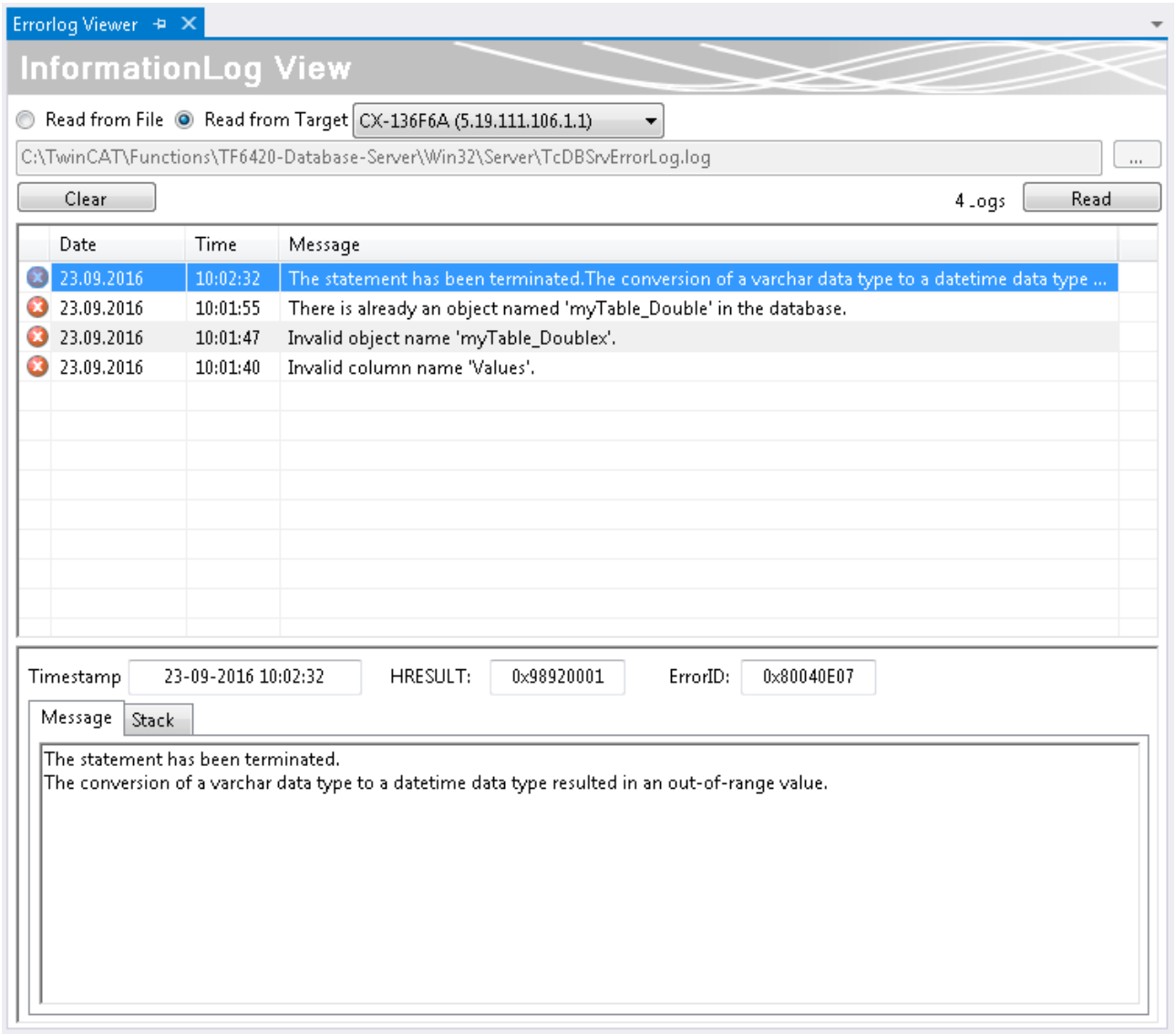


ID	Bezeichnung	Funktion
1	Zielsystem	Auswahl des Zielsystems mit installiertem TwinCAT Database Server
2	Start	Manueller Start des AutoLog-Modus
3	Einloggen	Einloggen auf den aktiven AutoLog-Prozess
4	Ausloggen	Ausloggen aus dem aktiven AutoLog-Prozess
5	Stopp	Manueller Stopp des AutoLog-Modus
6	Autologgruppen	Auflistung konfigurierter Autolog-Gruppen auf dem Zielsystem
7	Symbole	Auflistung konfigurierter Symbole der ausgewählten AutoLog-Gruppe

5.1.2.1.7 InformationLog View

Der InformationLog View ist ein Tool, um die Logdateien vom TwinCAT Database Server auszulesen. Protokollierte Informationen werden mit einem Zeitstempel, IDs und Fehlermeldungen im Klartext angezeigt.

Die Log-Dateien können nicht nur über den direkten Dateizugriff eingesehen oder geleert werden, sondern auch direkt über das Target. Gerade für verteilte Database Server im Netzwerk ist dies vorteilhaft, um einen schnellen und einfachen Zugriff auf die Logdatei zu erlangen. Für diesen Zugriff muss eine Route zum Zielgerät bestehen.



5.2 Datenbanken

Der TwinCAT Database Server ist das Bindeglied zwischen der TwinCAT SPS und Datenbanksystemen. Er unterstützt eine ganze Reihe von Datenbanken. Neben klassischen Datenbanken wie Microsoft SQL oder Oracle, können auch XML- und ASCII-Dateien als Datenbank verwendet werden. Im Bereich der ODBC Datenbanken ist es sogar möglich, Datenbank-Connection-Strings einzutragen, um mit Datenbanken zu kommunizieren, welche normalerweise nicht unterstützt werden.

Die beiden nachfolgenden Tabellen zeigen in einer Übersicht welche Datenbanken auf welchen Betriebssystem-Plattformen unterstützt werden und welche Datenbanken für den Daten-Export und -Import für das TwinCAT Scope zur Verfügung stehen.

Plattformunterstützung

Übersicht, welche Datenbankverbindungen von welcher Plattform unterstützt werden.

Datenbank	Windows		Windows CE		TwinCAT/BSD	
	Lokal	Remote	Lokal	Remote	Lokal	Remote
MS SQL	X	X	-	X	-	X
MS SQL Compact	X	-	X	-	-	-
MySQL	X	X	-	X*	X	X
Oracle DB	X	X	-	-	-	-
SQLite	X	-	X**	-	X*	-
ASCII-File	X	-	X	-	X	-
XML	X	-	X	-	X	-
ODBC	X*	X*	-	-	X*	X*
MS Access	X*	-	-	-	-	-
MS Excel	X*	-	-	-	-	-
MongoDB	X	X	-	-	X	X
PostgreSQL	X	X	-	-	-	X
InfluxDB 1.7 | 1.8	X	X	-	-	X	X
InfluxDB 2	X	X	-	-	-	X

*es müssen zusätzliche Server- bzw. Client-Treiber für die Datenbank auf dem Gerät installiert werden

**gilt nur für Geräte mit ARM-Architektur

TwinCAT Scope Unterstützung

Übersicht, welche Datenbanken für den Datenimport und -Export im TwinCAT Scope unterstützt werden. Das TwinCAT Scope arbeitet dabei immer mit dem TwinCAT Database Server zusammen.

Datenbank	Scope Export	Scope Import
MS SQL	X	X
MS SQL Compact	-	X
MySQL	-	X
Oracle DB	-	X
SQLite	-	X
ASCII-File	X	X
XML	-	X
ODBC	-	X
MS Access	-	X
MS Excel	-	X
MongoDB	-	-
PostgreSQL	-	X
InfluxDB 1.7 | 1.8	-	X
InfluxDB 2	-	X

Wie die einzelnen Datenbanken konfiguriert und die Datensätze in die SPS „gemappt“ werden, wird in den nachfolgenden Abschnitten erläutert.

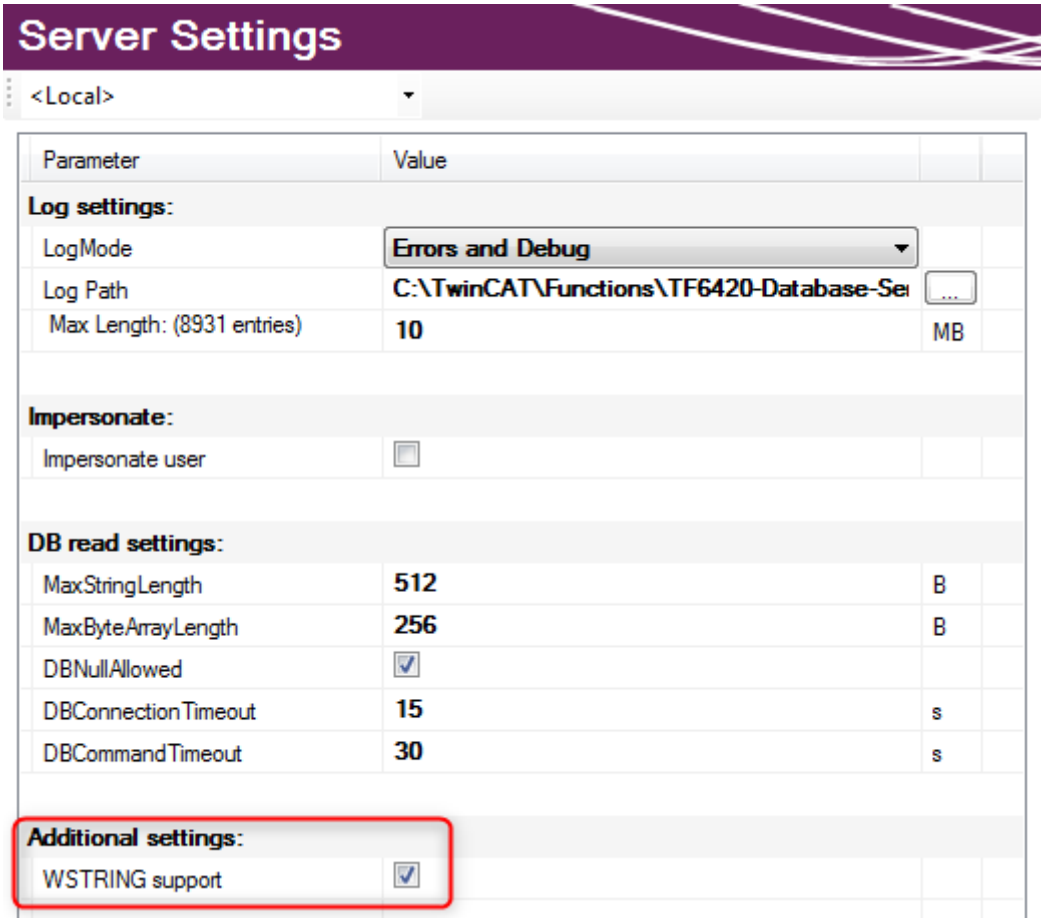
5.2.1 Allgemeine Informationen

Auf den folgenden Seiten finden Sie einige allgemeine Informationen zu den unterstützten Datenbanken. Diese Informationen sind datenbankübergreifend und behandeln Themen wie Netzwerk-Zugriffe, Datentyp-Unterstützung und Betriebssystem-Unterstützung.

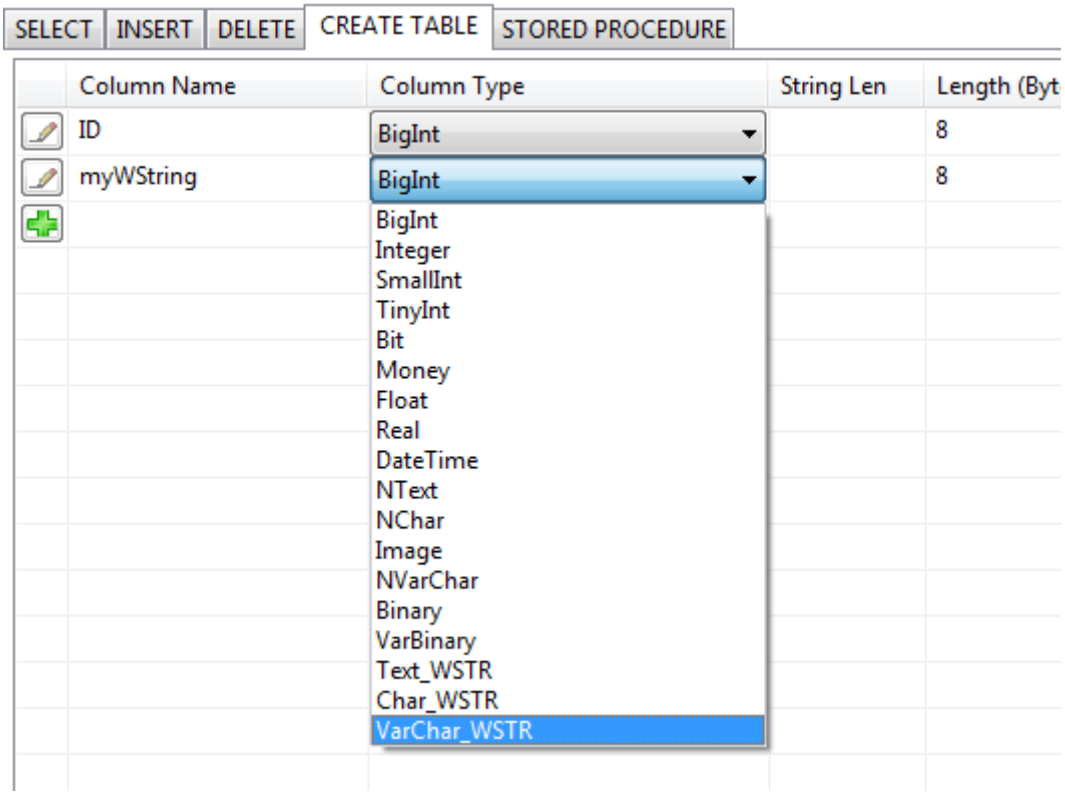
5.2.1.1 WString-Unterstützung

Um den Unicode-Zeichensatz zu verwenden, steht der WSTRING zur Verfügung.

Damit dieser Datentyp mit dem TwinCAT Database Server über den PLC Expert Mode in die Datenbank geschrieben werden kann, muss er in den Server-Einstellungen aktiviert werden.



Dieser Datentyp benötigt pro Zeichen 2 Byte. Beachten Sie dies bei der Erstellung der Tabellenstruktur. Um diese in der Datenbank im UTF16-Format abspeichern zu können, muss die Spalte dem Zeichensatz entsprechend angelegt sein. Dafür kann auch der SQL Query Editor verwendet werden.



Folgende Datenbanken werden mit dem Database Server unterstützt:

Datenbank	UTF8	UTF16	Zeichensatzfestlegung	Performance-Beeinträchtigung
MySQL	x	x	spaltenspezifisch	x
MSSQL	x	x	spaltenspezifisch	
Oracle	x	x	spaltenspezifisch	
PostgreSQL	x	x	datenbankübergreifend	x
Andere	x			

Performance-Beeinträchtigung

Bei einigen Datenbanken kommt es zu Performance-Beeinträchtigungen, da zusätzliche SQL-Befehle verschickt werden müssen, um den Zeichensatz auszulesen.

Die WString-Unterstützung ist ab der Version 3.1.31.4 vorhanden.

5.2.1.2 Bulk-Unterstützung

Der TwinCAT Database Server unterstützt für eine Auswahl von Datenbanken auch sogenannte BULK-Befehle. BULK-Befehle sind SQL-Anweisungen, welche gesammelte Daten in mehrere Zeilen einer Tabelle einfügt. Die Verwendung von BULK Insert Befehlen führt in der Regel zu einer besseren Leistung, als Prozesse die für jede hinzuzufügende Zeile eine einzelne Insert-Anweisung zur Datenbank schicken.

Aktuell werden BULK-Befehle über den FB_PLCDBCmdEvt Baustein von dem TwinCAT Database Server für Microsoft SQL Datenbanken unterstützt.

Beispiel-Kommando: 'SQLBULK<INSERT>#MyTable‘

5.2.2 MS SQL Datenbank

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von Microsoft SQL Datenbanken.

Kompatible Versionen: Microsoft SQL Datenbank 20xx.

Deklaration im TwinCAT Database Server Konfigurator

Mircosoft SQL Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „Microsoft SQL Server“ aus.
Provider	„SQLOLEDB“ oder der Provider des SQL Native Clients z. B. „SQLNCLI10“
Server	Geben Sie hier den Namen ihres SQL-Servers an. Beispiel: „TESTSERVER\SQLEXPRESS“
Database	Tragen Sie hier den Namen der Datenbank ein. Wenn die Datenbank noch nicht existiert, kann sie mit dem Create-Button erzeugt werden. Entsprechende Berechtigungen müssen vorhanden sein.
Authentication	Option, um sich mit einem bestimmten Nutzer an der Datenbank anzumelden.
Username	Geben Sie hier den Benutzernamen an.
Passwort	Tragen Sie hier das zum Benutzer passende Passwort ein.

Windows-CE-Unterstützung

Diese Datenbank wird auch von der Windows-CE-Variante des TwinCAT Database Servers unterstützt. Die Anbindung kann nicht lokal, aber über eine Netzwerkverbindung erfolgen.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	MS SQL	TwinCAT PLC
BigInt	bigint	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	integer	DINT
SmallInt	smallint	INT
TinyInt	tinyint	SINT
Bit_	bit	BYTE
Money	money	LREAL
Float	float	LREAL
Real_	real	REAL
DateTime	datetime	DT
NText	ntext	STRING
NChar	nchar	STRING
Image	image	ARRAY OF BYTE
NVarChar	nvarchar	STRING
Binary	binary	ARRAY OF BYTE
VarBinary	varbinary	ARRAY OF BYTE

Datentyp-Unterstützung

Diese Datenbank unterstützt den Datentyp WSTRING. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.2.1 Hinweise zum Microsoft SQL Server

Logs im Windows Eventlog

Error Event „Report Server Windows Service (SQLEXPRESS) kann nicht mit der Berichtsserver-Datenbank verbunden werden.“	Stoppen Sie im SQL Configuration Manager unter SQL Server 2005 den SQL Server Reporting Services (SQLEXPRESS) und setzen Sie den **Start Mode** auf „Manual“. Der Reporting Service wird nicht vom Database Server benötigt.
Information Event „'TcDataLogger'-Datenbank wird gestartet“	Öffnen Sie im SQL Server Management Studio Express im Bereich **Datenbanken/TcDataLogger** über das Kontextmenü die **Eigenschaften** und setzen Sie unter **Optionen** die Option **Automatisch schließen** auf FALSE. 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. Dann werden keine Events mehr geloggt. Zwischen den unterschiedlichen Eventtypen kann nicht unterschieden werden.

Wählen Sie im SQL Configuration Manager unter SQL Server 2005 Services den SQL-Server aus (SQLEXPRESS) und öffnen Sie über das Kontextmenü die **Eigenschaften**. In der Registerkarte **Advanced** gibt es den Unterpunkt **Startup Parameter**. Die einzelnen Parameter sind mit Semikolon getrennt. Fügen Sie den Parameter „-n“ hinzu und starten Sie den Service neu.

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

5.2.3 MS SQL Compact Datenbank

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von Microsoft SQL Compact Datenbanken. Die MS SQL Compact ist eine ideale Datenbank für Embedded-Anwendungen. Sie hat einen geringen Ressourcenbedarf und stellt dennoch die notwendige Funktionalität für relationale Datenbanken bereit.

Kompatible Versionen: Microsoft SQL Compact Datenbank 3.5

Deklaration im TwinCAT Database Server Konfigurator

Mircosoft SQL Compact Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „Microsoft Compact SQL“ aus.
Database URL	Tragen Sie hier den Namen und den Pfad der Datenbank ein. Wenn die Datenbank noch nicht existiert, kann sie mit dem Create-Button erzeugt werden. Entsprechende Berechtigungen müssen vorhanden sein.
Authentication	Option, um sich mit einem Passwort an der Datenbank anzumelden.
Passwort	Tragen Sie hier das Passwort ein.

Windows-CE-Unterstützung

Diese Datenbank wird auch von der Windows-CE-Variante des TwinCAT Database Servers unterstützt. Die Anbindung kann lokal erfolgen.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	MS SQL Compact	TwinCAT PLC
BigInt	bigint	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	integer	DINT
SmallInt	smallint	INT
TinyInt	tinyint	SINT
Bit_	bit	BYTE
Money	money	LREAL
Float	float	LREAL
Real_	real	REAL
DateTime	datetime	DT
NText	ntext	STRING
NChar	nchar	STRING
Image	image	ARRAY OF BYTE
NVarChar	nvarchar	STRING
Binary	binary	ARRAY OF BYTE
VarBinary	varbinary	ARRAY OF BYTE

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.4 MySQL Datenbank

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von MySQL Datenbanken.

Deklaration im TwinCAT Database Server Konfigurator

MySQL Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „MySQL“ aus.
Server	Geben Sie hier den Namen oder die IP-Adresse Ihres Servers an.
Database	Tragen Sie hier den Namen der Datenbank ein. Wenn die Datenbank noch nicht existiert, kann sie mit dem Create-Button erzeugt werden. Entsprechende Berechtigungen müssen vorhanden sein.
Port	Geben Sie hier den Port für die Kommunikation mit der MySQL an. Default: 3306.
Username	Geben Sie hier den Nutzernamen an.
Passwort	Tragen Sie hier das zum Nutzer passende Passwort ein.

Windows-CE-Unterstützung

Diese Datenbank wird auch von der Windows-CE-Variante des TwinCAT Database Servers unterstützt. Die Anbindung kann nicht lokal, aber über eine Netzwerkverbindung erfolgen.

Die Installation der nötigen Komponenten erfolgt **nicht** automatisch und müssen manuell installiert werden. Dafür muss **zusätzlich** die Bibliothek „MySql.Data.CF.dll“ in der Version 6.7.8.0 auf das CE-Gerät in den TwinCAT Database Server–Ordner kopiert werden. *(\Hard Disk\TwinCAT\Functions\TF6420-Database-Server\Server)*

Diese Bibliothek für CE unterstützt Verbindungen mit einem MySQL-Server bis einschließlich Version 8.0.28.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	MySQL	TwinCAT PLC
BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	INT	DINT
SmallInt	SMALLINT	INT
TinyInt	TINYINT	SINT
Bit_	CHAR(1)	STRING
Money	DEZIMAL(18,4)	LREAL
Float	DOUBLE	LREAL
Real_	FLOAT	REAL
DateTime	DATETIME	DT
NText	TEXT	STRING
NChar	CHAR	STRING
Image	BLOB	ARRAY OF BYTE
NVarChar	VARCHAR	STRING
Binary	BLOB	ARRAY OF BYTE
VarBinary	BLOB	ARRAY OF BYTE

Treiber für „Big-Windows“ Systeme

The used MySQLConnector is under MIT licence and under Copyright:

Copyright (c) 2016 Bradley Grainger

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Datentyp-Unterstützung

Diese Datenbank unterstützt den Datentyp WSTRING. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.5 Oracle Datenbank

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von Oracle Datenbanken. Für die Anbindung an die Oracle Datenbank wird der sogenannte ODP-Treiber eingesetzt.

Kompatible Versionen: Oracle 9i, 10g und 11g, sowie höher

Der TwinCAT Database Server benötigt die 32-bit-Version der .NET-ODP-Komponenten.

Deklaration im TwinCAT Database Server Konfigurator

Oracle Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „Oracle ODP“ aus.
Host	Geben Sie hier die IP oder den Hostnamen der Datenbank an.
Service Name	Hier wird der Name des Services oder der Datenbankname angegeben.
Port	Geben Sie hier optional den Kommunikations-Port an. Default: 1521.
Protocol	Geben Sie hier optional das Protokoll an. Default: TCPIP.
Schema	Hier wird das genutzte Schema der Datenbank angegeben.
Username	Geben Sie hier den Benutzernamen an.
Passwort	Tragen Sie das zum Benutzer passende Passwort ein.

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	Oracle	TwinCAT PLC
BigInt	DECIMAL(15,0)	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	INTEGER	DINT
SmallInt	SMALLINT	INT
TinyInt	SMALLINT	SINT
Bit_	CHAR(1)	BYTE
Money	DECIMAL(18,4)	LREAL
Float	DOUBLE PRECISION	LREAL
Real_	FLOAT	REAL
DateTime	DATE	DT
NText	VARCHAR(254)	STRING
NChar	CHAR(254)	STRING
Image	BLOB	ARRAY OF BYTE
NVarChar	NVARCHAR(254)	STRING
Binary	BLOB	ARRAY OF BYTE
VarBinary	BLOB	ARRAY OF BYTE

Datentyp-Unterstützung

Diese Datenbank unterstützt den Datentyp WSTRING. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.6 SQLite

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von SQLite Datenbanken. Die SQLite ist eine ideale Datenbank für Embedded-Anwendungen. Diese dateibasierte SQL-Datenbank benötigt keine Installation, sondern ist schon im TwinCAT Database Server integriert. Die relationale Datenbank bietet die meisten Funktionen von SQL-Datenbanken und unterstützt die Befehle des SQL92-Standards. Mit dieser Datenbank ist eine zuverlässige und schnelle Speicherung der Daten möglich. Eine Nutzerunterscheidung ist mit der Datenbank jedoch nicht möglich. Deshalb eignet sie sich besonders gut zur gesicherten Speicherung von Variablen auf dem lokalen System.

Deklaration im TwinCAT Database Server Konfigurator

SQLite Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „SQLite“ aus.
SQLite Database File	Tragen Sie hier Namen und Pfad der Datenbank ein. Sie können auch den Browser-Dialog nutzen. Wenn die Datenbank noch nicht existiert, kann sie mit dem Create-Button erzeugt werden. Entsprechende Berechtigungen müssen vorhanden sein.
Authentication	Eine Option, um sich mit einem bestimmten Benutzer an der Datenbank anzumelden.
Passwort	Tragen Sie hier das zum Benutzer passende Passwort ein.

Windows-CE-Unterstützung

Diese Datenbank wird von der Windows-CE-Variante des TwinCAT Database Servers ausschließlich auf Geräten mit ARM-Prozessor unterstützt. Die Anbindung kann lokal erfolgen.

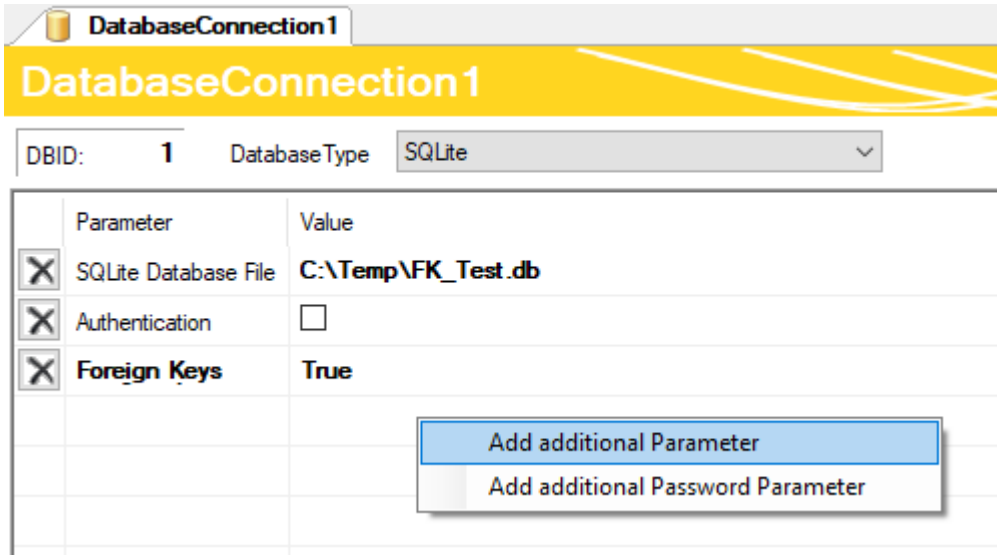
TwinCAT/BSD-Unterstützung

Diese Datenbank wird vom TwinCAT Database Server auf TwinCAT/BSD unterstützt. Zusätzlich wird für den Einsatz jedoch die Installation des Package „sqlite3“ vom Package Repository vorausgesetzt.

Nutzung von Primär / Fremdschlüssel

Um Primär- und Fremdschlüssel in SQLite Datenbanken verwenden zu können, muss in der Konfiguration ein zusätzlicher Parameter angegeben werden.

„Foreign Keys“ -> „True“



Datentyp-Mapping zwischen DB und SPS

SQLite verfügt über fünf interne Basisdatentypen. Für genauere Interpretationen der Daten werden zusätzlich weitere Datentypen unterstützt, welche in der Dokumentation des Datenbankherstellers aufgeführt sind.

E_ColumnTypes	SQLite	TwinCAT PLC
BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	INT	DINT
SmallInt	SMALLINT	INT
TinyInt	TINYINT	BYTE
Bit_	BOOLEAN	BOOL
Money	DOUBLE	LREAL
Float	FLOAT	LREAL
Real_	REAL	REAL
DateTime	DATETIME	DT
NText	TEXT	STRING
NChar	NCHAR	STRING
Image	BLOB	ARRAY OF BYTE
NVarChar	NVARCHAR	STRING
Binary	BLOB	ARRAY OF BYTE
VarBinary	BLOB	ARRAY OF BYTE

Besonderheit: String bzw. Binary Datentypen sind in Sqlite unbegrenzt. Der TwinCAT 3 Database Server benötigt jedoch feste Begrenzungen, welche in den allgemeinen Server Einstellungen eingestellt werden können.

Inkompatible Datentypen

Es ist möglich, dass durch Software von Drittanbietern inkompatible Datentypbezeichnungen in der Datenbank angelegt werden, die der TwinCAT3 Database Server nicht interpretieren kann. Hier ist es hilfreich, den SQL Query Builder zu nutzen.

Datentyp-Unterstützung

Diese Datenbank unterstützt den Datentyp WSTRING. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.7 ASCII-File

Hinweise zur Konfiguration von ASCII-Dateien als Datenbanken. Die Datei wird vom TwinCAT Database Server automatisch erzeugt. Ein Create Database ist nicht notwendig. Die erstellte Datei kann in anderen Tabellenkalkulationsprogrammen, wie z. B. Microsoft Excel importiert und weiter verarbeitet werden.

Deklaration im TwinCAT Database Server Konfigurator

ASCII File als Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „ASCII“ aus.
ASCII File	Hier wird der Pfad zur ASCII-Datei angegeben. Die Datei wird automatisch vom TwinCAT Database Server erzeugt.
Value Separator	Hier können Sie den Separator für die Werte, also für die Spalten, angeben. Default: „;“
Old ASCII-DB Format	Schalten Sie hier optional aus Kompatibilitätsgründen auf das alte ASCII-Format der TwinCAT Database Server Versionen 3.0.x um. Verwenden Sie dort die Standard-Tabellenstruktur.
DBValue Type	Nur aktiv, wenn **Old ASCII-DB Format** aktiviert ist. Sie können zwischen BYTES und DOUBLE wählen. Bei DOUBLE sind die Werte im Klartext, bei BYTES ein Byte-Stream.

Nicht unterstützte Funktionen

Die automatische ID-Generierung wird von dieser Datenbank nicht unterstützt. Falls im Configure Mode die Standardtabellenstruktur genutzt wird, wird der Wert der ID nicht gesetzt.

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

Windows-CE-Unterstützung

Diese Datenbank wird auch von der Windows-CE-Variante des TwinCAT Database Servers unterstützt. Die Anbindung kann lokal erfolgen.

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.8 XML-Database

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von XML-Dateien als Datenbanken. Aufbau der Datenbank, Tabellen und Spalten werden in einer XSD-Datei definiert. Die XML, die XSD und eine XSL mit Style-Informationen werden durch ein **Create** mit dem TwinCAT Database Server Konfigurator erzeugt. Aufgrund der XSL-Datei können Sie die XML in einem Web-Browser öffnen und so eine grafisch aufbereitete Ansicht der Datenbank bzw. der Tabelle erhalten.

Weitere tiefergehende Informationen zum Arbeiten mit XML-Dateien als Datenbank finden Sie im Abschnitt „XML - Informationen [▶ 139]“.

Deklaration im TwinCAT Database Server Konfigurator

XML Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „XML“ aus.
XML Database File	Tragen Sie hier Namen und Pfad der XML-Datei ein.
XML Schema File	Tragen Sie hier Namen und Pfad der XSD-Datei ein.
Database	Tragen Sie hier den Namen der Datenbank ein. Wenn die Datenbank noch nicht existiert, kann sie mit dem Create-Button erzeugt werden. Entsprechende Berechtigungen müssen vorhanden sein. Im Fall von XML Datenbanken werden die XML, die XSD und die XSL automatisch erzeugt.

Windows-CE-Unterstützung

Diese Datenbank wird auch von der Windows-CE-Variante des TwinCAT Database Servers unterstützt. Die Anbindung kann lokal erfolgen.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	XML	TwinCAT PLC
BigInt	bigint	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	integer	DINT
SmallInt	smallint	INT
TinyInt	tinyint	BYTE
Bit_	bit	BOOL
Money	money	LREAL
Float	float	LREAL
Real_	real	LREAL
DateTime	datetime	DT
NText	ntext	STRING
NChar	nchar	STRING
Image	image	ARRAY OF BYTE
NVarChar	nvarchar	STRING
Binary	binary	ARRAY OF BYTE
VarBinary	varbinary	ARRAY OF BYTE

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.8.1 XML - Informationen

1. XML-Datei verwenden als Datenbank mit dem TwinCAT 3 Database Server

2. XPath Querries auf eine XML-Datei anwenden mit dem TwinCAT 3 Database Server

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

1. XML als Datenbank

XSD-Schema für Standard-Tabellenstruktur:

```
<?xmlversion="1.0"?>
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:simpleTypename="bigint">
<xsd:restrictionbase="xsd:long" />
</xsd:simpleType>
<xsd:simpleTypename="datetime">
<xsd:restrictionbase="xsd:dateTime" />
</xsd:simpleType>
<xsd:simpleTypename="ntext_80">
<xsd:restrictionbase="xsd:string">
<xsd:maxLengthvalue="80" />
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleTypename="float">
<xsd:restrictionbase="xsd:double" />
</xsd:simpleType>
<xsd:complexTypename="myTable_Double_Type">
<xsd:sequence>
<xsd:elementminOccurs="0"maxOccurs="unbounded"name="row">
<xsd:complexType>
<xsd:attributename="ID"type="bigint" />
<xsd:attributename="Timestamp"type="datetime" />
<xsd:attributename="Name"type="ntext_80" />
<xsd:attributename="Value" type="float" />
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:elementname="TestDB_XML">
<xsd:complexType>
<xsd:sequenceminOccurs="1"maxOccurs="1">
<xsd:elementname="myTable_Double"type="myTable_Double_Type" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

XML-Datei für Standard-Tabellenstruktur (Beispiel):

```
<?xmlversion="1.0"encoding="UTF-8"?>
<TestDB_XMLxmlns:xs="http://www.w3.org/2001/XMLSchema-
nstance"xs:noNamespaceSchemaLocation="TestDB_XML.xsd">
<myTable_Double>
<rowID="1"Timestamp="2012-03-08T12:45:08"Name="TestValue1"Value="222.222" />
<rowID="2"Timestamp="2012-03-08T12:45:14"Name="TestValue1"Value="222.222" />
<rowID="3"Timestamp="2012-03-08T12:45:18"Name="TestValue1"Value="222.222" />
<rowID="4"Timestamp="2012-03-08T12:45:22"Name="TestValue1"Value="222.222" />
```

```
<rowID="5"Timestamp="2012-03-08T12:45:23"Name="TestValue1"Value="222.222" />
</myTable_Double>
</TestDB_XML>
```

Datatypes für XML-Tabellen:

```
<xsd:simpleTypename="bigint">
<xsd:restrictionbase="xsd:long" />
</xsd:simpleType>

<xsd:simpleTypename="datetime">
<xsd:restrictionbase="xsd:dateTime" />
</xsd:simpleType>

<xsd:simpleTypename="ntext_80"> //Länge kann individuell angegeben werden
<xsd:restrictionbase="xsd:string">
<xsd:maxLengthvalue="80" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleTypename="float">
<xsd:restrictionbase="xsd:double" />
</xsd:simpleType>

<xsd:simpleTypename="binary_1"> //Länge kann individuell angegeben werden
<xsd:restrictionbase="xsd:hexBinary">
<xsd:maxLengthvalue="1" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleTypename="bit">
<xsd:restrictionbase="xsd:boolean" />
</xsd:simpleType>

<xsd:simpleTypename="image_1"> //Länge kann individuell angegeben werden
<xsd:restrictionbase="xsd:hexBinary">
<xsd:maxLengthvalue="1" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleTypename="integer">
<xsd:restrictionbase="xsd:int" />
</xsd:simpleType>

<xsd:simpleTypename="money">
<xsd:restrictionbase="xsd:double" />
</xsd:simpleType>

<xsd:simpleTypename="nchar_50"> //Länge kann individuell angegeben werden
<xsd:restrictionbase="xsd:string">
<xsd:maxLengthvalue="50" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleTypename="nvarchar_50"> //Länge kann individuell angegeben
werden
<xsd:restrictionbase="xsd:string">
<xsd:maxLengthvalue="50" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleTypename="real">
<xsd:restrictionbase="xsd:double" />
</xsd:simpleType>

<xsd:simpleTypename="smallint">
<xsd:restrictionbase="xsd:short" />
</xsd:simpleType>

<xsd:simpleTypename="tinyint">
<xsd:restrictionbase="xsd:byte" />
</xsd:simpleType>

<xsd:simpleTypename="varbinary_1"> //Länge kann individuell angegeben werden
<xsd:restrictionbase="xsd:hexBinary">
<xsd:maxLengthvalue="1" />
</xsd:restriction>
</xsd:simpleType>
```

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	XML	TwinCAT PLC
BigInt	bigint	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	integer	DINT
SmallInt	smallint	INT
TinyInt	tinyint	BYTE
Bit_	bit	BOOL
Money	money	LREAL
Float	float	LREAL
Real_	real	LREAL
DateTime	datetime	DT
NText	ntext	STRING
NChar	nchar	STRING
Image	image	ARRAY OF BYTE
NVarChar	nvarchar	STRING
Binary	binary	ARRAY OF BYTE
VarBinary	varbinary	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 INTO myTable_Double (ID, Timestamp, Name, Value) VALUES(1, CURRENT_TIMESTAMP, 'TestValue1' , 1234.5678)
- INSERT INTO myTable_Double (Timestamp, Name) VALUES(CURRENT_TIMESTAMP, 'TestValue1');
- INSERT INTO myTable_Double VALUES(1, CURRENT_TIMESTAMP, 'TestValue1', 1234.5678);
- INSERT INTO myTable_Double VALUES(1, '2010-01-06 12:13:14', 'TestValue1', 1234.5678);

Beispiele für unterstützte SELECT-Befehle:

- SELECTID, Timestamp, Name, Value FROM myTable_Double;
- SELECT* FROM myTable_Double;
- SELECTTimestamp, 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

2. XML-Standard-XPath-Funktion

XPath-Typen

Die Syntax der Präfixe der XPaths im TwinCAT Database Server ist folgende:
XPATH_[Type]<[Position]>#[Path]

Es gibt 4 verschiedene Typen des XPath:

- SEL
  - Liest Daten aus der XML und liefert sie an die SPS zurück
- ADD
  - Fügt die übergebenen Daten der XML an der gewählten Position an.
- UPD
  - Ersetzt die vorhandene Information der XML an der gewählten Position mit den neuen Daten.
- DEL
  - Löscht die Daten in der XML an der gewählten Position.

Für die Positionen stehen 3 verschiedene Angaben zur Verfügung:

- ATTR
  - Betrifft alle Attributwerte vom ausgewählten XML-Tag.
- TAG
  - Betrifft den InnerText-Wert des ausgewählten XML-Tag.
- SUBTAG
  - Betrifft die InnerText-Werte aller SubTags des ausgewählten XML-Tag.
  - 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.

Beispiele:

XML-Datei:

```
<?xmlversion="1.0"encoding="utf-8" ?>
<TestXML>
<Nodeattr1="1"attr2="Node1">
<SubNode1>SubNodeWert1</SubNode1>
<SubNode2>200</SubNode2>
<SubNode3>SubNodeWert3</SubNode3>
<SubNode4>400.5</SubNode4>
<SubNode5>SubNodeWert5</SubNode5>
</Node>
<Nodeattr1="2"attr2="Node2">
<SubNode1>SubNodeWert1</SubNode1>
<SubNode2>200</SubNode2>
<SubNode3>SubNodeWert3</SubNode3>
<SubNode4>400.5</SubNode4>
<SubNode5>SubNodeWert5</SubNode5>
</Node>
</TestXML>
```

XML-Schema:

```
<?xmlversion="1.0"encoding="utf-8"?>
<xs:schemaattributeFormDefault="unqualified"elementFormDefault="qualified"xmlns:xs="http://
www.w3.org/2001/XMLSchema">
<xs:elementname="TestXML">
<xs:complexType>
<xs:sequence>
<xs:elementmaxOccurs="unbounded"name="Node">
<xs:complexType>
<xs:sequence>
<xs:elementname="SubNode1"type="xs:string" />
<xs:elementname="SubNode2"type="xs:short" />
```

```
<xs:elementname="SubNode3"type="xs:string" />
<xs:elementname="SubNode4"type="xs:double" />
<xs:elementname="SubNode5"type="xs:string" />
</xs:sequence>
<xs:attributename="attr1" type="xs:integer"use="required" />
<xs:attributename="attr2" type="xs:string"use="required" />
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Beispiel für XPATH<ATTR>

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

Zurückgelieferte Struktur, wenn **kein** Schema vorhanden ist:

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

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

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

Beispiel für XPATH<TAG>

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

Zurückgelieferter Wert, wenn **kein** Schema vorhanden ist: SubNode2 : T_MaxString := '200';

Zurückgelieferter Wert, wenn **ein** Schema vorhanden ist: SubNode2 : INT := 200;

Beispiel für XPATH<SUBTAG>

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

Zurückgelieferte Struktur, wenn **kein** Schema vorhanden ist:

```
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 ist:

```
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
```

Besonderheit unter der Verwendung des **FB_PLCDBCmd**:

Anders als bei der üblichen Implementierung der FB_PLCDBCmd werden über die eingestellten Parameter (ST_ExpParameter) nicht die Platzhalter für die jeweilige Anweisung angegeben, sondern das Schema der übergebenen oder zurückgelieferten Daten.

Unterstützte Funktionsbausteine

- FB_DBRecordSelect
- FB_DBRecordSelect_EX
- FB_DBRecordArraySelect

5.2.9 ODBC Datenbanken

Viele Datenbanken bieten ODBC-Schnittstellen. Der TwinCAT Database Server verfügt ebenfalls über diese Schnittstelle. Daher kann im TwinCAT Database Konfigurator ganz allgemein eine ODBC_Database im Datenbank-Konfigurationsmenü ausgewählt werden. Im sogenannten „Free Connection String [▶ 144]“ können Sie eigene Connection Strings formen, indem Sie mit **Add additional parameter** den String aufbauen.

Weitere bekannte und häufig genutzte ODBC-Datenbanken sind als Template verfügbar. Dazu gehören:

- MySQL [▶ 145]
- Oracle [▶ 146]
- PostgreSQL [▶ 147]
- IBM DB2 [▶ 148]
- Firebird [▶ 150]

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

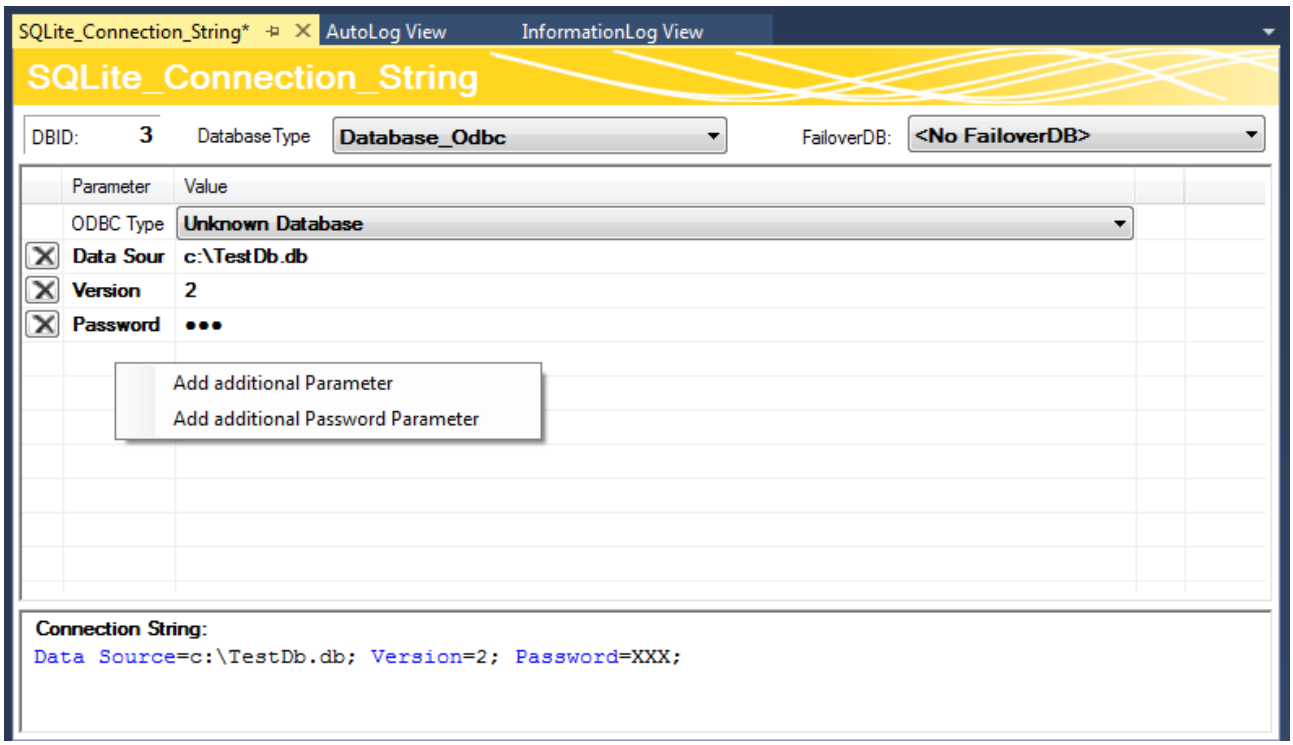
5.2.9.1 Free Connection String

Wenn Sie eine Datenbank mit ODBC-Schnittstelle benutzen wollen, welche vom TwinCAT Database Server standardmäßig nicht unterstützt wird, können Sie als **ODBC Type** „Unknown Database“ auswählen.

Deklaration im TwinCAT Database Server Konfigurator

ODBC Free Connection String Datenbank	
Database Type	„Odbc_Database“
ODBC Type	Wählen Sie im Drop-down-Menü „Unknown Database“ aus.

Suchen Sie einfach den Connection String für die ODBC Datenbank heraus und bauen Sie ihn im Konfigurationsfenster des TwinCAT Database Servers nach.



Für den Nachbau stehen Ihnen im Kontextmenü zwei Kommandos zur Verfügung:

- **Add additional Parameter**
  Fügt einen allgemeinen Parameter für den Connection String ein. Dieser kann beliebig benannt werden, so wie es der Connection String vorgibt.
- **Add additional Password Parameter**
  Fügt einen speziellen Passwort-Parameter ein, dessen Wert im Konfigurator und in der Konfigurationsdatei unkenntlich bzw. verschlüsselt ist.

Funktionsweise mit Free Connection String

Damit der TwinCAT Database Server mit einem Free Connection String arbeiten kann, muss der entsprechende Treiber auf dem Zielsystem des TwinCAT Database Servers installiert sein. Es kann nur der „SQL Expert Mode [▶ 18]“ verwendet werden!

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.9.2 MySQL

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von MySQL Datenbanken mit ODBC.

Deklaration im TwinCAT Database Server Konfigurator

ODBC MySQL Datenbank	
Database Type	„Odbc_Database“
ODBC Type	Wählen Sie im Drop-down-Menü „MySQL aus.
Driver	Tragen Sie hier den tatsächlich installierten Treiber ein.
Server	Geben Sie hier den Namen oder die IP-Adresse ihres Servers an.
Database	Tragen Sie hier den Namen der Datenbank eintragen.
Port	Geben Sie hier den Port für die Kommunikation mit der MySQL an. Default: 3306.
Option	Default: 2 „Return matched rows instead of affected rows“
Uid	Geben Sie hier Benutzernamen an.
Pwd	Tragen Sie hier das zum Benutzer passende Passwort ein.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	MySQL	TwinCAT PLC
BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	INT	DINT
SmallInt	SMALLINT	INT
TinyInt	TINYINT	SINT
Bit_	CHAR(1)	STRING
Money	DEZIMAL(18,4)	LREAL
Float	DOUBLE	LREAL
Real_	FLOAT	REAL
DateTime	DATETIME	DT
NText	TEXT	STRING
NChar	CHAR	STRING
Image	BLOB	ARRAY OF BYTE
NVarChar	VARCHAR	STRING
Binary	BLOB	ARRAY OF BYTE
VarBinary	BLOB	ARRAY OF BYTE

Datentyp-Unterstützung

Diese Datenbank unterstützt den Datentyp WSTRING. (Siehe WString-Unterstützung [▶ 128])

Funktionsweise

Alle Funktionsweisen des TwinCAT Database Servers können auf die ODBC Templates angewendet werden. Dies gilt nicht für den „Free Connection String [▶ 144]“.

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.9.3 Oracle

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von Oracle Datenbanken mit ODBC.

Deklaration im TwinCAT Database Server Konfigurator

ODBC Oracle Datenbank	
Database Type	„Odbc_Database“
ODBC Type	Wählen Sie im Drop-down-Menü „Oracle“ aus.
Driver	Stellen Sie hier den tatsächlich installierten Treiber ein.
Server	Geben Sie hier den Namen oder die IP-Adresse ihres Servers an.
Uid	Geben Sie hier den Benutzernamen an.
Pwd	Tragen Sie hier das zum Benutzer passende Passwort ein.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	Oracle	TwinCAT PLC
BigInt	DECIMAL(15,0)	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	INTEGER	DINT
SmallInt	SMALLINT	INT
TinyInt	SMALLINT	SINT
Bit_	CHAR(1)	BYTE
Money	DECIMAL(18,4)	LREAL
Float	DOUBLE PRECISION	LREAL
Real_	FLOAT	REAL
DateTime	DATE	DT
NText	VARCHAR(254)	STRING
NChar	CHAR(254)	STRING
Image	BLOB	ARRAY OF BYTE
NVarChar	NVARCHAR(254)	STRING
Binary	BLOB	ARRAY OF BYTE
VarBinary	BLOB	ARRAY OF BYTE

Datentyp-Unterstützung

Diese Datenbank unterstützt den Datentyp WSTRING. (Siehe WString-Unterstützung [▶ 128])

Funktionsweise

Alle Funktionsweisen des TwinCAT Database Servers können auf die ODBC Templates angewendet werden. Dies gilt nicht für den „Free Connection String [▶ 144]“.

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.9.4 PostgreSQL

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von PostgreSQL Datenbanken mit ODBC.

Deklaration im TwinCAT Database Server Konfigurator

ODBC PostgreSQL Datenbank	
Database Type	„Odbc_Database“
ODBC Type	Wählen Sie im Drop-down-Menü „PostgreSQL“ aus.
Driver	Stellen Sie hier den tatsächlich installierten Treiber ein.
Server	Geben Sie hier den Namen oder die IP-Adresse ihres Servers an.
Database	Tragen Sie hier den Namen der Datenbank ein.
Port	Geben Sie hier den Port für die Kommunikation mit der PostgreSQL an. Default: 5432.
Uid	Geben Sie hier den Benutzernamen an.
Pwd	Tragen Sie hier das zum Benutzer passende Passwort ein.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	PostgreSQL	TwinCAT PLC
BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	integer	DINT
SmallInt	smallint	INT
TinyInt	smallint	INT
Bit_	bit	BYTE
Money	money	LREAL
Float	Double precision	LREAL
Real_	real	REAL
DateTime	timestamp	DT
NText	text	STRING
NChar	character	STRING
Image	byte	ARRAY OF BYTE
NVarChar	Character varying	STRING
Binary	byte	ARRAY OF BYTE
VarBinary	byte	ARRAY OF BYTE

Datentyp-Unterstützung

Diese Datenbank unterstützt den Datentyp WSTRING. Der Zeichensatz muss bei Erstellung der Datenbank eingerichtet werden.

Funktionsweise

Alle Funktionsweisen des TwinCAT Database Servers können auf die ODBC Templates angewendet werden. Dies gilt nicht für den „Free Connection String [▶ 144]“.

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.9.5 IBM DB2

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von IBM DB2 Datenbanken mit ODBC.

Deklaration im TwinCAT Database Server Konfigurator

ODBC IBM DB2 Datenbank	
Database Type	„Odbc_Database“
ODBC Type	Wählen Sie im Drop-down-Menü „IBM DB2“ aus.
Driver	Tragen Sie hier den tatsächlich installierten Treiber eintragen.
Hostname	Geben Sie hier den Namen oder die IP-Adresse ihres Servers an.
Database	Tragen Sie hier den Namen der Datenbank ein.
Port	Geben Sie hier den Port für die Kommunikation mit der IBM DB2 an. Default: 50000.
Protocol	Default: TCPIP
Uid	Geben Sie hier den Benutzernamen an.
Pwd	Tragen Sie hier das zum Benutzer passende Passwort ein.
LONGDATACOMPAT	Default: 1

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	IBM DB2	TwinCAT PLC
BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	INT	DINT
SmallInt	SMALLINT	INT
TinyInt	SMALLINT	INT
Bit_	VARCHAR(1)	STRING(1)
Money	DEZIMAL(18,4)	LREAL
Float	DOUBLE PRECISION	LREAL
Real_	FLOAT	LREAL
DateTime	TIMESTAMP	DT
NText	LONG VARCHAR	STRING
NChar	CHAR(254)	STRING
Image	BLOB	ARRAY OF BYTE
NVarChar	NVARCHAR(254)	STRING
Binary	BLOB	ARRAY OF BYTE
VarBinary	BLOB	ARRAY OF BYTE

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

Funktionsweise

Alle Funktionsweisen des TwinCAT Database Servers können auf die ODBC Templates angewendet werden. Dies gilt nicht für den „Free Connection String [▶ 144]“.

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.9.6 Firebird

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von Firebird Datenbanken mit ODBC.

Deklaration im TwinCAT Database Server Konfigurator

ODBC Firebird Datenbank	
Database Type	„Odbc_Database“
ODBC Type	Wählen Sie im Drop-down-Menü „Firebird“ aus.
Driver	Stellen Sie hier den tatsächlich installierten Treiber ein.
Database	Tragen Sie hier den Namen der Datenbank ein.
Client	
Uid	Geben Sie hier den Benutzernamen an.
Pwd	Tragen Sie hier das zum Benutzer passende Passwort ein.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	Firebird	TwinCAT PLC
BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	INTEGER	DINT
SmallInt	SMALLINT	INT
TinyInt	TINYINT	INT
Bit_	CHAR(1)	STRING
Money	DEZIMAL(18,4)	LREAL
Float	FLOAT	REAL
Real_	DOUBLE PRECISION	LREAL
DateTime	TIMESTAMP	DT
NText	VARCHAR(254)	STRING
NChar	CHAR(254)	STRING
Image	BLOB	ARRAY OF BYTE
NVarChar	VARCHAR(254)	STRING
Binary	BLOB	ARRAY OF BYTE
VarBinary	BLOB	ARRAY OF BYTE

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

Funktionsweise

Alle Funktionsweisen des TwinCAT Database Servers können auf die ODBC Templates angewendet werden. Dies gilt nicht für den „Free Connection String [▶ 144]“.

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.10 MS 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.

Deklaration im TwinCAT Database Server Konfigurator

Microsoft Access Datenbank	
DBValueType	Wollen Sie nur alphanummerische Datentypen loggen und Boolean, wählen Sie „Double“. Wollen Sie auch Strukturen und Strings mitloggen, wählen Sie „Bytes“.
DBType	Wählen Sie “MS Access”. 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“.
DBUrl	DBUrl enthält den Pfad zu der MDB-Datei. Z. B. *C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.mdb*
DBTable	DBTable enthält den Namen der Tabelle.

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

TwinCAT/BSD-Unterstützung

Diese Datenbank wird auf der Plattform TwinCAT/BSD nicht vom TwinCAT Database Server unterstützt.

Datentyp-Mapping zwischen DB und SPS

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

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.11 MS Excel Datenbank

Die Variablenwerte werden in einer Microsoft Excel Datenbank gespeichert.

Deklaration im TwinCAT Database Server Konfigurator

Microsoft Excel Datenbank	
DBValueType	Wollen Sie nur Alphanummerische Datentypen loggen und Boolean, wählen Sie „Double“. Wollen Sie auch Strukturen und Strings mitloggen, wählen Sie „Bytes“.
DBType	Wählen Sie „MS Excel“. PLC: eDBType_MSExcel.
DBServer	Wird nicht benötigt.
DBProvider	„Microsoft.Jet.OLEDB.4.0“ oder „Microsoft.ACE.OLEDB.12.0“
DBUrl	DBUrl enthält den Pfad zu der Excel-Datei. Z. B. *C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.xls*
DBTable	DBTable enthält den Namen der Tabelle.

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

TwinCAT/BSD-Unterstützung

Diese Datenbank wird auf der Plattform TwinCAT/BSD nicht vom TwinCAT Database Server unterstützt.

Datentyp-Mapping zwischen DB und SPS

E_DBColumnTypes	MS Excel	PLC Control
eDBColumn_BigInt	Number	LREAL
eDBColumn_Integer	Number	LREAL
eDBColumn_SmallInt	Number	LREAL
eDBColumn_TinyInt	Number	LREAL
eDBColumn_Bit	BOOLEAN	BOOL
eDBColumn_Money	Currency	LREAL
eDBColumn_Float	Number	LREAL
eDBColumn_Real	Number	LREAL
eDBColumn_DateTime	Date	DT
eDBColumn_NText	Text	STRING(255)
eDBColumn_NChar	Text	STRING(255)
eDBColumn_NVarChar	Text	STRING(255)

Nicht unterstützte Funktionen

Die automatische ID-Generierung wird von dieser Datenbank nicht unterstützt. Falls im Configure Mode die Standardtabellenstruktur genutzt wird, wird der Wert der ID nicht gesetzt.

Nicht unterstützte Datentypen

Binary, VarBinary und Image werden mit Excel Datenbanken nicht unterstützt.

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.12 MongoDB

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von MongoDB Datenbanken.

Deklaration im TwinCAT Database Server Konfigurator

MongoDB	
Database Type	Wählen Sie im Drop-down-Menü „MongoDb“ aus.
Server	Geben Sie hier den Namen Ihres MongoDB-Servers an.
Database	Tragen Sie hier den Namen der Datenbank ein. Wenn die Datenbank noch nicht existiert, wird sie beim ersten Zugriff erzeugt.
Authentication	**None:** keine Authentifizierung **Username/Password:** Anmeldung mit Benutzernamen und Passwort **x509 Certificate:** Benutzername: ID des Users des Zertifikats Certificate Authority: Pfad zum signierenden Zertifikat (*.crt) Client Certificate: Pfad zum Client-Zertifikat (*.pfx) Client Private Key: Passwort für das Client Zertifikat **GSSAPI/Kerberos:** Anmeldung mit Benutzernamen und Passwort **LDAP(PLAIN):** Anmeldung mit Benutzernamen und Passwort (Da Benutzername und Passwort im Klartext übertragen werden, wird diese Möglichkeit nicht empfohlen)

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

Datentyp-Mapping zwischen DB und SPS

MongoDB	TwinCAT PLC
long	LINT
int	DINT
bool	BYTE
double	LREAL
timestamp	DT
string	STRING
binData	ARRAY OF BYTE
objectId	T_ObjectId_MongoDB
array	ARRAY
object	STRUCT

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

MongoDB im PLC Expert/Configure Mode

Der PLC Expert und der Configure Mode verwenden in ihren Abläufen das vorgegebene Schema einer Datenbank. Im Normalfall wird sich das Schema der verwendeten Strukturen im laufenden Betrieb nicht ändern. Um dennoch die Bausteine nutzen zu können, benötigt der TwinCAT 3 Database Server eine Beschreibung des Tabellenschemas. Für MongoDB wird deshalb eine Tabelle simuliert.
Verwenden Sie im SQL Query Editor den Reiter **SQL**, sowie die Unterkategorie **CREATE TABLE** um eine Tabelle bzw. in diesem Fall eine Collection zu erstellen. Zusätzlich wird, anders als bei relationalen Datenbanken, ein Eintrag in eine Metadaten-Collection erzeugt. Hier sind Informationen zum Tabellenschema für den TwinCAT 3 Database Server abgespeichert.

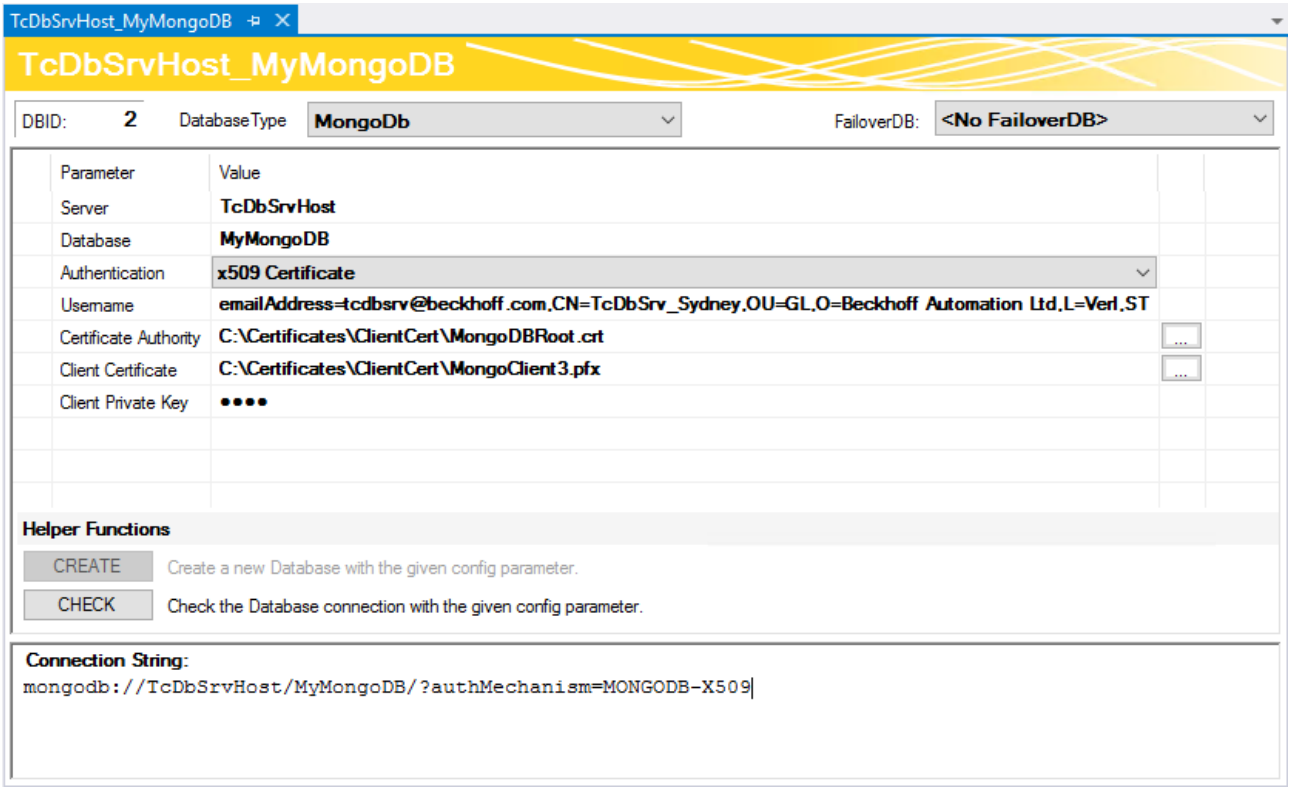
Um erweiterte Funktionsmöglichkeiten zu nutzen, wie z.B. Strukturen beliebiger Hierarchie oder flexible Datensätze, wird empfohlen die NoSQL-Funktionsbausteine zu verwenden.

Verwendung von Zertifikaten

Mit der MongoDB ist unter anderem die Unterstützung der Authentifizierung mittels Zertifikaten möglich. Dazu wird unter Authentication die Methode ‚x509 Certificate' ausgewählt. Es erscheinen folgende Felder:

Username	Nutzername des betreffenden Zertifikats
Certificate Authority	Pfad zum SSL-Zertifikat der Zertifizierungsstelle. Dies kann auch ein selbst signiertes Zertifikat sein.
Client Certificate	Client Zertifikat welches vom SSL-Zertifikat signiert wurde.
Client Certificate Password	Passwort des Client Zertifikats.

Datenbankverbindung zur MongoDB über Zertifikate konfigurieren:



HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.13 PostgreSQL

In diesem Abschnitt finden Sie Hinweise zu Konfiguration und Nutzung der Datenbank PostgreSql. PostgreSQL ist eine objektrelationale Open Source Datenbank mit einer Client-Server Infrastruktur. Der TwinCAT 3 Database Server nutzt zur Verbindung die Npgsql API.

Deklaration im TwinCAT Database Server Konfigurator

PostgreSql Datenbank	
Database Type	Wählen Sie im Drop-down-Menü „PostgreSql“ aus.
Server	Name oder IP des Datenbank-Server
Database	Name der Datenbank auf dem Server
Port	Port der Datenbank
Authentication	Authentifizierungsmethode der Datenbank
Username	Geben Sie hier den Benutzernamen an.
Passwort	Tragen Sie hier das zum Benutzer passende Passwort ein.
Client Certificate	Pfad zum genutzten Client-Zertifikat (.pfx)
Client Certificate Password	Passwort des referenzierten Client Zertifikats

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	PostgreSQL	TwinCAT PLC
BigInt	Bigint	T_ULARGE_INTEGER (TcUtilities.lib)
Integer	Integer	DINT
SmallInt	Smallint	INT
TinyInt		SINT
Bit_	Bit	BYTE
Money	Money	LREAL
Float	Double precision	LREAL
Real_	Real	REAL
DateTime	Timestamp without time zone	DT
NText	Text	STRING
NChar	Character	STRING
Image		ARRAY OF BYTE
NVarChar	Character varying	STRING
Binary	Bytea	ARRAY OF BYTE
VarBinary		ARRAY OF BYTE

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

5.2.14 InfluxDB

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von InfluxDB.

Unterstützte Version: 1.7.x, 1.8.x

Deklaration im TwinCAT Database Server Konfigurator

InfluxDB	
Database Type	Wählen Sie im Drop-down-Menü „InfluxDB“ aus.
Server	Geben Sie hier die Adresse zum gewünschen Datenbank-Server an.
Database	Tragen Sie hier den Namen der Datenbank ein. Wenn die Datenbank noch nicht existiert, können sie diese mit „Create“ erzeugen.
Authentication	**None:** keine Authentifizierung **Username/Password:** Anmeldung mit Benutzernamen und Passwort

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	InfluxDB	TwinCAT PLC
BigInt	Integer	LINT
Integer	Integer	DINT
SmallInt	Integer	INT
TinyInt	Integer	BYTE
Bit_	Boolean	BOOL
Money	Float	LREAL
Float	Float	LREAL
Real_	Float	LREAL
DateTime		DT
NText	String	STRING
NChar	String	STRING
Image		ARRAY OF BYTE
NVarChar	String	STRING
Binary		ARRAY OF BYTE
VarBinary		ARRAY OF BYTE
	Tag	STRING
	„time“	LINT

Zeit

InfluxDB besitzt als Zeitreihendatenbank einige Besonderheiten. Jedes Measurement (Tabelle) einer Series beinhaltet die Zeitspalte, sogenannte Tag-Spalten und Field-Spalten. Die Zeitspalte wird in der Datenbank als UNIX-Epochzeit abgespeichert. Die Funktionsbausteine des Database Server arbeiten mit der TwinCAT Zeit (Anzahl von 100ns Schritten seit 01.01.1601). Diese werden in die UNIX-Epoch Zeit konvertiert. Der FB_SQLDBCommand wird von dieser Konvertierung ausgenommen. Hier können eigene freie Zeitstempel ohne Konvertierung übergeben werden. Die Precision ist dabei auf „ns“-Genauigkeit eingestellt. Zeiten sind in InfluxDB als ID zusammen mit den Tag-Spalten eindeutig. Falls die Tags und die Zeit gleich sind, wird ein Datensatz überschrieben.

Standardtabellenstruktur

Die Standardtabellenstruktur für die InfluxDB sieht in der SPS wie folgt aus:

ColumnName	InfluxDB	TwinCAT PLC
time	Integer	LINT
Name	Tag	T_MaxString
Value	Float	LREAL

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

5.2.15 InfluxDB2

In diesem Abschnitt finden Sie Hinweise zur Konfiguration und zum Datentyp-Mapping von InfluxDB2.

Unterstützte Version: 2.x

Deklaration im TwinCAT Database Server Konfigurator

InfluxDB	
Database Type	Wählen Sie im Drop-down-Menü „InfluxDB2“ aus.
Server	Geben Sie hier die Adresse zum gewünschen Datenbank-Server an.
Database	Tragen Sie hier den Namen der Datenbank ein. Wenn die Datenbank noch nicht existiert, können sie diese mit „Create“ erzeugen.
Authentication	**None:** keine Authentifizierung **Username/Password:** Anmeldung mit Benutzernamen und Passwort
Token	Zugangstoken für den Zugriff auf die Datenbank. Dieser kann mit den erforderlichen Rechten in der Datenbanksoftware erstellt werden.

Windows-CE-Unterstützung

Diese Datenbank wird unter Windows CE nicht vom TwinCAT Database Server unterstützt.

Datentyp-Mapping zwischen DB und SPS

E_ColumnTypes	InfluxDB2	TwinCAT PLC
BigInt	Integer	LINT
Integer	Integer	DINT
SmallInt	Integer	INT
TinyInt	Integer	BYTE
Bit_	Boolean	BOOL
Money	Float	LREAL
Float	Float	LREAL
Real_	Float	LREAL
DateTime	DateTime	DT
NText	String	STRING
NChar	String	STRING
Image	-	ARRAY OF BYTE
NVarChar	String	STRING
Binary	-	ARRAY OF BYTE
VarBinary	-	ARRAY OF BYTE
Tag	Tag	STRING
BigInt	„_time“	LINT

Zeit

InfluxDB2 besitzt als Zeitreihendatenbank einige Besonderheiten. Jedes Measurement (Tabelle) einer Series beinhaltet die Zeitspalte, sogenannte Tag-Spalten und Field-Spalten. Die Zeitspalte wird in der Datenbank als UNIX-Epochzeit abgespeichert. Die Funktionsbausteine des Database Server arbeiten mit der TwinCAT Zeit (Anzahl von 100ns Schritten seit 01.01.1601). Diese werden in die UNIX-Epoch Zeit konvertiert. Der FB_SQLDBCommand wird von dieser Konvertierung ausgenommen. Hier können eigene freie Zeitstempel ohne Konvertierung übergeben werden. Die Precision ist dabei auf „ns“-Genauigkeit eingestellt. Zeiten sind in InfluxDB als ID zusammen mit den Tag-Spalten iieindeutig. Falls die Tags und die Zeit gleich sind, wird ein Datensatz überschrieben.

Datentyp-Unterstützung

WSTRING wird von dieser Datenbank nicht unterstützt. (Siehe WString-Unterstützung [▶ 128])

HINWEIS

Datensicherheit

Bei Flash-Speichermedien ist die Anzahl der Schreibzugriffe begrenzt. Die Flash-Speichermedien können ausfallen und Datenverlust droht.

- Erstellen Sie regelmäßig Backups von Ihrem System. Benutzen Sie die IPC-Diagnose, um den Status der Flash-Speichermedien zu ermitteln.

6 SPS-API

6.1 Tc3_Database

6.1.1 Funktionsbausteine

Die Funktionsbausteine der Tc3_Database.compiled-library gliedern sich in Anlehnung an das Grundkonzept [▶ 18] in drei Bereiche:

- Configure Mode:
  Hier sind Bausteine enthalten, um das Lesen und Schreiben von im Konfigurator definierten AutoLog-Gruppen zu steuern.
- PLC Expert Mode:
  Hier sind Bausteine für klassische SPS-Programmierer zusammengefasst.
- SQL Expert Mode:
  Mit diesen Bausteinen können IT- und SPS-Experten mit erweitertem Datenbankwissen selbst SQL-Kommandos in der SPS zusammenbauen.
- NoSql Expert Mode:
  Mit diesen Bausteinen können IT- und SPS-Experten mit erweitertem Datenbankwissen über NoSql-Datenbanken Kommandos erstellen und an die Datenbank schicken.

Nutzung des Tc3_EventLoggers

Der TwinCAT 3 Database Server unterstützt die Tc3_Eventlogger API. Weitere Informationen finden Sie unter Unterstützung des Tc3_EventLogger [▶ 223]

6.1.1.1 Configure Mode

6.1.1.1.1 FB_ConfigTcDBSrvEvt

Funktionsbaustein zum Erstellen, Auslesen und Löschen von Konfigurationseinträgen für den TwinCAT Database Server.

Syntax

Definition:

```
FUNCTION_BLOCK FB_ConfigTcDBSrvEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage;
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Create [▶ 161]	Lokal	Erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server
Read [▶ 162]	Lokal	Liest die aktuelle Konfiguration des TwinCAT Database Servers
Delete [▶ 163]	Lokal	Löscht die Datenbank und die AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.1.1.1 Create

Diese Methode erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server. Optional kann ein neuer Eintrag auch nur temporär vom TwinCAT Database Server verwendet werden. Er wird dann nicht in die XML-Datei geschrieben.

Syntax

```
METHOD Create : BOOL
VAR_INPUT
    pTcDBSrvConfig: POINTER TO BYTE;
    cbTcDBSrvConfig: UDINT;
    bTemporary: BOOL := TRUE;
    pConfigID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pTcDBSrvConfig	POINTER TO BYTE	Zeiger der Konfigurationsstruktur, die erzeugt werden soll.
cbTcDBSrvConfig	UDINT	Länge der Konfigurationsstruktur
bTemporary	BOOL	Gibt an, ob die Konfiguration in die XML-Datei gespeichert werden soll.
pConfigID	POINTER TO UDINT	Rückgabezeiger der Konfigurations-ID (hDBID oder hAutoLogGrpID)

Das Anlegen von Autolog-Gruppen wird derzeit nicht unterstützt.

Rückgabewert

Name	Typ	Beschreibung
Create	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    // Any other ConfigType can be used here
    stConfigDB      : T_DBConfig_MsCompactSQL;
    tcMessage       : I_TcMessage;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Recipes.sdf';

IF fbConfigTcDBSrv.Create(
    pTcDBSrvConfig:= ADR(stConfigDB),
    cbTcDBSrvConfig:= SIZEOF(stConfigDB),
    bTemporary:= TRUE,
    pConfigID:= ADR(myConfigHandle))
THEN
    IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.1.1.2 Read

Mit dieser Methode können die aktuellen Konfigurationen des TwinCAT Database Servers gelesen werden. Wenn temporäre Konfigurationen dabei sind, werden diese entsprechend markiert.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    pDBConfig: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    cbDBConfig: UDINT;
    pAutoLogGrpConfig: POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF
ST_ConfigAutoLogGrp;
    cbAutoLogGrpConfig: UDINT;
    pDBCount: POINTER TO UDINT;
    pAutoLogGrpCount: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 249]] OF ST_ConfigDB [▶ 232]	Pointer-Adresse des Arrays, in das die Datenbankkonfigurationen geschrieben werden sollen.
cbDBConfig	UDINT	Länge des Datenbankkonfiguration-Arrays
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS [▶ 249]] OF ST_ConfigAutoLogGrp [▶ 231]	Pointer-Adresse des Arrays, in das die AutoLogGrp Konfigurationen geschrieben werden sollen.
cbAutoLogGrpConfig	UDINT	Länge des AutoLogGrp-Konfiguration-Arrays
pDBCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der Datenbankkonfigurationen gespeichert werden.
pAutoLogGrpCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der AutoLogGrp-Konfigurationen gespeichert werden.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv   : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    aDBConfig         : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    aAutoGrpConfig    : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp;
    nDbCount          : UDINT;
    nAutoGrpCount     : UDINT;
    tcMessage         : I_TcMessage;
END_VAR
```

```
IF fbConfigTcDBSrv.Read(
    pDBConfig := ADR(aDBConfig),
    cbDBConfig := SIZEOF(aDBConfig),
    pAutologGrpConfig := ADR(aAutoGrpConfig),
    cbAutoLogGrpConfig := SIZEOF(aAutoGrpConfig),
    pDBCount := ADR(nDbCount),
    pAutoLogGrpCount := ADR(nAutoGrpCount))
 THEN
    IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.1.1.3 Delete

Mit dieser Methode können Datenbank und AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers gelöscht werden.

Syntax

```
METHOD Delete : BOOL
VAR_INPUT
    eTcDBSrvConfigType: E_TcDBSrvConfigType;
    hConfigID: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
eTcDBSrvConfigType	E_TcDBSrvConfigType	Typ der zu löschenden Konfiguration (Datenbank / AutoLog-Gruppe)
hConfigID	UDINT	ID der zu löschenden Konfiguration (hDBID oder hAutoLogGrpID)

Rückgabewert

Name	Typ	Beschreibung
Delete	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    tcMessage       : I_TcMessage;
END_VAR
```

```
IF fbConfigTcDBSrv.Delete(
    eTcDBSrvConfigType := E_TcDBSrvConfigType.Database,
    hConfigID := myConfigHandle) THEN
IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.1.2 FB_PLCDBAutoLogEvt

Funktionsbaustein mit vier Methoden zum Starten und Stoppen von definierten AutoLog-Gruppen, sowie zum Auslesen des entsprechenden Status der Gruppe.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBAutoLogEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage;
    bBusy_Status: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist, mit Ausnahme der Status Methode.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.
bBusy_Status	BOOL	Die Methode Status kann unabhängig von den anderen drei Methoden des Bausteins ausgeführt werden und hat folglich ein eigenes Busy Flag. Ist TRUE, sobald die Methode Status aktiv ist.

Methoden

Name	Definitionsort	Beschreibung
RunOnce [▶ 165]	Lokal	Führt die AutoLog-Gruppe einmalig aus
Start [▶ 166]	Lokal	Startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen
Status [▶ 166]	Lokal	Fragt den Status der AutoLog-Gruppen ab.
Stop [▶ 167]	Lokal	Stoppt den AutoLog-Modus

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.1.2.1 RunOnce

Mit dieser Methode kann eine AutoLog-Gruppe einmalig ausgeführt werden. Zum Beispiel aufgrund eines Ereignisses in der Steuerung.

Syntax

```
METHOD RunOnce : BOOL
VAR_INPUT
    hAutoLogGrpID: UDINT;
    bAll: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hAutoLogGrpID	UDINT	ID der AutoLog-Gruppe, die einmalig ausgeführt werden soll.
bAll	BOOL	Wenn TRUE, werden alle AutoLog-Gruppen werden einmalig ausgeführt.

Rückgabewert

Name	Typ	Beschreibung
RunOnce	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.RunOnce(hAutologGrpID := 1, bAll := FALSE) THEN
    ; // ...
END_IF
```

6.1.1.1.2.2 Start

Diese Methode startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen.

Syntax

```
METHOD Start : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Start	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Start() THEN
    ; // ...
END_IF
```

6.1.1.1.2.3 Status

Mit dieser Methode kann der Status der AutoLog Gruppen abgefragt werden. Im Rumpf des Bausteins ist ein eigenes Busy Flag für diese Methode vorgesehen, da sie unabhängig von den anderen Methoden des Bausteins aufgerufen werden kann: bBusy_Status.

Syntax

```
METHOD Status : BOOL
VAR_INPUT
    tCheckCycle: TIME;
    pError: POINTER TO BOOL;
    pAutoLogGrpStatus: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
    cbAutoLogGrpStatus: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
tCheckCycle	TIME	Intervallzeit, in der das Statusarray aktualisiert wird.
pError	POINTER TO BOOL	TRUE, wenn ein Fehler beim AutoLog Modus aufgetreten ist.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 249]] OF ST_AutoLogGrpStatus [▶ 246]	Adresse zum Statusarray, welches alle Gruppen beinhaltet.
cbAutoLogStatus	UDINT	Länge des Statusarrays

Rückgabewert

Name	Typ	Beschreibung
Status	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt(sNetID:='', tTimeout := T#5S);
    bError            : BOOL;
    aAutologGrpStatus : ARRAY[0..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
END_VAR
```

```
IF fbPLCDBAutoLog.Status(tCheckCycle := T#30S, ADR(bError), ADR(aAutologGrpStatus), SIZEOF(aAutologGrpStatus)) THEN
    ; // ...
END_IF
```

6.1.1.1.2.4 Stop

Diese Methode stoppt den AutoLog-Modus.

Syntax

```
METHOD Stop : BOOL
```

Rückgabewert

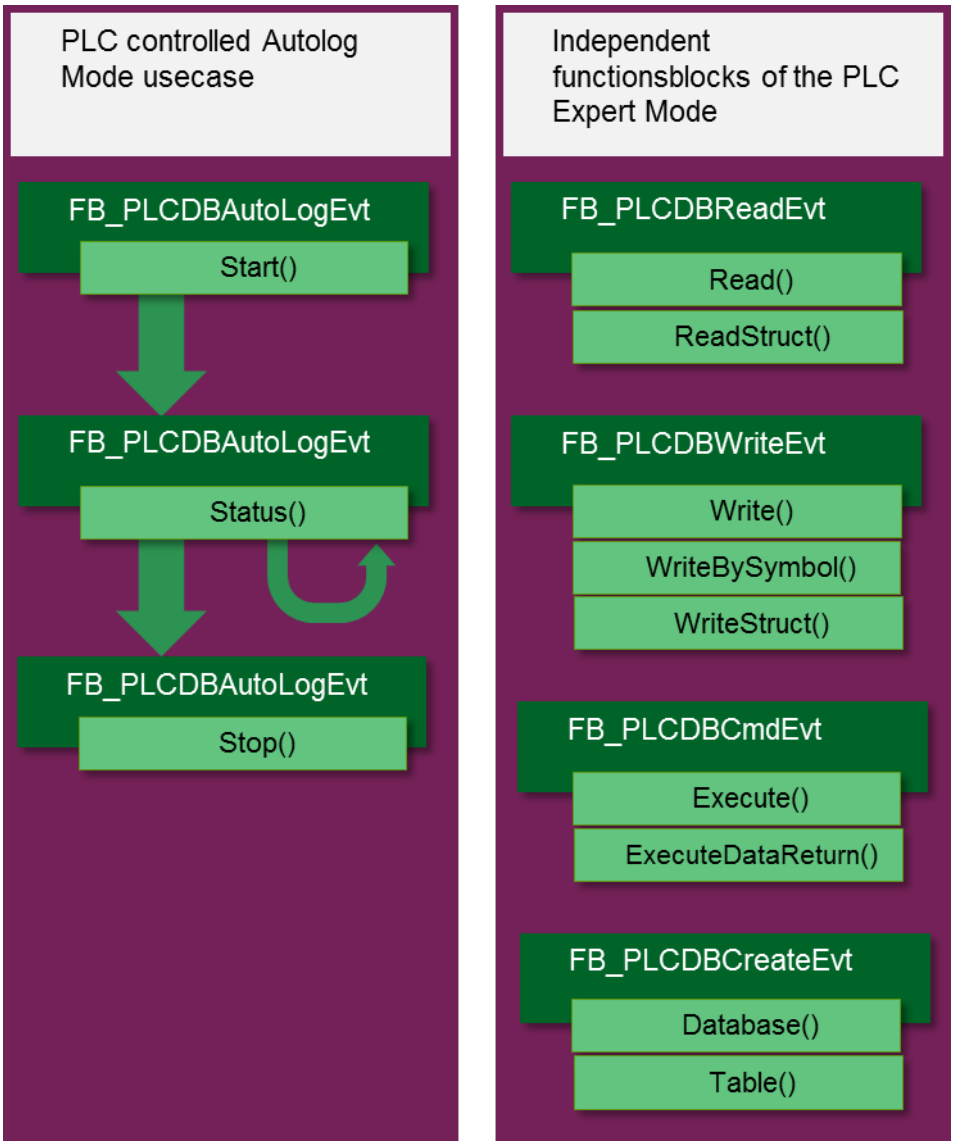
Name	Typ	Beschreibung
Stop	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Stop() THEN
    ; // ...
END_IF
```

6.1.1.2 PLC Expert Mode



6.1.1.2.1 FB_ConfigTcDBSrvEvt

Funktionsbaustein zum Erstellen, Auslesen und Löschen von Konfigurationseinträgen für den TwinCAT Database Server.

Syntax

Definition:

```
FUNCTION_BLOCK FB_ConfigTcDBSrvEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage;
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Create [▶ 169]	Lokal	Erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server
Read [▶ 170]	Lokal	Liest die aktuelle Konfiguration des TwinCAT Database Servers
Delete [▶ 171]	Lokal	Löscht die Datenbank und die AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.2.1.1 Create

Diese Methode erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server. Optional kann ein neuer Eintrag auch nur temporär vom TwinCAT Database Server verwendet werden. Er wird dann nicht in die XML-Datei geschrieben.

Syntax

```
METHOD Create : BOOL
VAR_INPUT
    pTcDBSrvConfig: POINTER TO BYTE;
    cbTcDBSrvConfig: UDINT;
    bTemporary: BOOL := TRUE;
    pConfigID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pTcDBSrvConfig	POINTER TO BYTE	Zeiger der Konfigurationsstruktur, die erzeugt werden soll.
cbTcDBSrvConfig	UDINT	Länge der Konfigurationsstruktur
bTemporary	BOOL	Gibt an, ob die Konfiguration in die XML-Datei gespeichert werden soll.
pConfigID	POINTER TO UDINT	Rückgabezeiger der Konfigurations-ID (hDBID oder hAutoLogGrpID)

Das Anlegen von Autolog-Gruppen wird derzeit nicht unterstützt.

Rückgabewert

Name	Typ	Beschreibung
Create	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    // Any other ConfigType can be used here
    stConfigDB      : T_DBConfig_MsCompactSQL;
    tcMessage       : I_TcMessage;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Recipes.sdf';

IF fbConfigTcDBSrv.Create(
    pTcDBSrvConfig:= ADR(stConfigDB),
    cbTcDBSrvConfig:= SIZEOF(stConfigDB),
    bTemporary:= TRUE,
    pConfigID:= ADR(myConfigHandle))
THEN
    IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.2.1.2 Read

Mit dieser Methode können die aktuellen Konfigurationen des TwinCAT Database Servers gelesen werden. Wenn temporäre Konfigurationen dabei sind, werden diese entsprechend markiert.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    pDBConfig: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    cbDBConfig: UDINT;
    pAutoLogGrpConfig: POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF
ST_ConfigAutoLogGrp;
    cbAutoLogGrpConfig: UDINT;
    pDBCount: POINTER TO UDINT;
    pAutoLogGrpCount: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 249]] OF ST_ConfigDB [▶ 232]	Pointer-Adresse des Arrays, in das die Datenbankkonfigurationen geschrieben werden sollen.
cbDBConfig	UDINT	Länge des Datenbankkonfiguration-Arrays
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS [▶ 249]] OF ST_ConfigAutoLogGrp [▶ 231]	Pointer-Adresse des Arrays, in das die AutoLogGrp Konfigurationen geschrieben werden sollen.
cbAutoLogGrpConfig	UDINT	Länge des AutoLogGrp-Konfiguration-Arrays
pDBCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der Datenbankkonfigurationen gespeichert werden.
pAutoLogGrpCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der AutoLogGrp-Konfigurationen gespeichert werden.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv   : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    aDBConfig         : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    aAutoGrpConfig    : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp;
    nDbCount          : UDINT;
    nAutoGrpCount     : UDINT;
    tcMessage         : I_TcMessage;
END_VAR
```

```
IF fbConfigTcDBSrv.Read(
    pDBConfig := ADR(aDBConfig),
    cbDBConfig := SIZEOF(aDBConfig),
    pAutologGrpConfig := ADR(aAutoGrpConfig),
    cbAutoLogGrpConfig := SIZEOF(aAutoGrpConfig),
    pDBCount := ADR(nDbCount),
    pAutoLogGrpCount := ADR(nAutoGrpCount))
 THEN
    IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.2.1.3 Delete

Mit dieser Methode können Datenbank und AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers gelöscht werden.

Syntax

```
METHOD Delete : BOOL
VAR_INPUT
    eTcDBSrvConfigType: E_TcDBSrvConfigType;
    hConfigID: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
eTcDBSrvConfigType	E_TcDBSrvConfigType	Typ der zu löschenden Konfiguration (Datenbank / AutoLog-Gruppe)
hConfigID	UDINT	ID der zu löschenden Konfiguration (hDBID oder hAutoLogGrpID)

Rückgabewert

Name	Typ	Beschreibung
Delete	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    tcMessage       : I_TcMessage;
END_VAR
```

```
IF fbConfigTcDBSrv.Delete(
    eTcDBSrvConfigType := E_TcDBSrvConfigType.Database,
    hConfigID := myConfigHandle) THEN
IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.2.2 FB_PLCDBAutoLogEvt

Funktionsbaustein mit vier Methoden zum Starten und Stoppen von definierten AutoLog-Gruppen, sowie zum Auslesen des entsprechenden Status der Gruppe.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBAutoLogEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage;
    bBusy_Status: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist, mit Ausnahme der Status Methode.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.
bBusy_Status	BOOL	Die Methode Status kann unabhängig von den anderen drei Methoden des Bausteins ausgeführt werden und hat folglich ein eigenes Busy Flag. Ist TRUE, sobald die Methode Status aktiv ist.

Methoden

Name	Definitionsort	Beschreibung
RunOnce [▶ 173]	Lokal	Führt die AutoLog-Gruppe einmalig aus
Start [▶ 174]	Lokal	Startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen
Status [▶ 174]	Lokal	Fragt den Status der AutoLog-Gruppen ab.
Stop [▶ 175]	Lokal	Stoppt den AutoLog-Modus

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.2.2.1 RunOnce

Mit dieser Methode kann eine AutoLog-Gruppe einmalig ausgeführt werden. Zum Beispiel aufgrund eines Ereignisses in der Steuerung.

Syntax

```
METHOD RunOnce : BOOL
VAR_INPUT
    hAutoLogGrpID: UDINT;
    bAll: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hAutoLogGrpID	UDINT	ID der AutoLog-Gruppe, die einmalig ausgeführt werden soll.
bAll	BOOL	Wenn TRUE, werden alle AutoLog-Gruppen werden einmalig ausgeführt.

Rückgabewert

Name	Typ	Beschreibung
RunOnce	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.RunOnce(hAutologGrpID := 1, bAll := FALSE) THEN
    ; // ...
END_IF
```

6.1.1.2.2.2 Start

Diese Methode startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen.

Syntax

```
METHOD Start : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Start	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Start() THEN
    ; // ...
END_IF
```

6.1.1.2.2.3 Status

Mit dieser Methode kann der Status der AutoLog Gruppen abgefragt werden. Im Rumpf des Bausteins ist ein eigenes Busy Flag für diese Methode vorgesehen, da sie unabhängig von den anderen Methoden des Bausteins aufgerufen werden kann: bBusy_Status.

Syntax

```
METHOD Status : BOOL
VAR_INPUT
    tCheckCycle: TIME;
    pError: POINTER TO BOOL;
    pAutoLogGrpStatus: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
    cbAutoLogGrpStatus: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
tCheckCycle	TIME	Intervallzeit, in der das Statusarray aktualisiert wird.
pError	POINTER TO BOOL	TRUE, wenn ein Fehler beim AutoLog Modus aufgetreten ist.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 249]] OF ST_AutoLogGrpStatus [▶ 246]	Adresse zum Statusarray, welches alle Gruppen beinhaltet.
cbAutoLogStatus	UDINT	Länge des Statusarrays

Rückgabewert

Name	Typ	Beschreibung
Status	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt(sNetID:='', tTimeout := T#5S);
    bError            : BOOL;
    aAutologGrpStatus : ARRAY[0..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
END_VAR
```

```
IF fbPLCDBAutoLog.Status(tCheckCycle := T#30S, ADR(bError), ADR(aAutologGrpStatus), SIZEOF(aAutologGrpStatus)) THEN
    ; // ...
END_IF
```

6.1.1.2.2.4 Stop

Diese Methode stoppt den AutoLog-Modus.

Syntax

```
METHOD Stop : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Stop	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLogEvt (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Stop() THEN
    ; // ...
END_IF
```

6.1.1.2.3 FB_PLCDBCreateEvt

Funktionsbaustein mit zwei Methoden. Mit der einen Methode können Datenbanken aus der SPS heraus auf einem in der SPS angegebenen Datenbank-Server erstellt werden. Mit der anderen Methode kann in einer angegebenen Datenbank eine neue Tabelle erzeugt werden.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBCreateEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
```

```
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Database [▶ 176]	Lokal	Erzeugt eine neue Datenbank
Table [▶ 177]	Lokal	Erzeugt eine neue Tabelle mit der Tabellenstruktur, die über ein Array mit x Elementen bzw. x Spalten in der SPS definiert wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.2.3.1 Database

Diese Methode erzeugt eine neue Datenbank. Optional kann angegeben werden, ob die erzeugte Datenbank auch für die Konfiguration des TwinCAT Database Servers übernommen werden soll.

Syntax

```
METHOD Database : BOOL
VAR_INPUT
    pDatabaseConfig: POINTER TO BYTE;
    cbDatabaseConfig: UDINT;
    bCreateXMLConfig: BOOL;
    pDBID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDatabaseConfig	POINTER TO BYTE	Adresse zu der Datenbankkonfigurationsstruktur [▶ 232]
cbDatabaseConfig	UDINT	Länge der Datenbankkonfigurationsstruktur
bCreateXMLConfig	BOOL	Gibt an, ob die neu erzeugte Datenbank als neuer Konfigurationseintrag in die XML-Datei geschrieben werden soll.
pDBID	UDINT	Gibt die hDBID zurück, wenn ein neuer Konfigurationseintrag erzeugt wurde.

Rückgabewert

Name	Typ	Beschreibung
Database	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBCreate : FB_PLCDBCreateEvt(sNetID := '', tTimeout := T#5S);
    stConfigDB    : T_DBConfig_MsCompactSQL;
    hDBID         : UDINT;
    tcMessage     : I_TcMessage;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Test.sdf';

IF fbPLCDBCreate.Database(
    pDatabaseConfig:= ADR(stConfigDB),
    cbDatabaseConfig := SIZEOF(stConfigDB),
    bCreateXMLConfig := TRUE,
    pDBID := ADR(hDBID))
THEN
    IF fbPLCDBCreate.bError THEN
        tcMessage := fbPLCDBCreate.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.2.3.2 Table

Diese Methode erzeugt eine neue Tabelle mit der Tabellenstruktur, die über ein Array mit x Elementen bzw. x Spalten in der SPS definiert wird.

Syntax

```
METHOD Table : BOOL
VAR_INPUT
    hDBID : UDINT;
    sTableName : T_MaxString;
    pTableCfg : POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ColumnInfo;
    cbTableCfg : UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	MaxString	Name der Tabelle, die erzeugt werden soll.
pTableCfg	POINTER TO ARRAY[0..MAX_DBCOLUMNS [▶ 249]] OF ST_ColumnInfo [▶ 247]	Gibt die Pointer-Adresse des Tabellenstrukturarrays an. In diesem Array werden die einzelnen Spalten beschrieben.
cbTableCfg	UDINT	Gibt die Länge der Arrays an, in dem die Spalten konfiguriert sind.

Rückgabewert

Name	Typ	Beschreibung
Table	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBCreate : FB_PLCDBCreateEvt(sNetID := '', tTimeout := T#5S);
    ColumnInfo    : ARRAY [0..14] OF ST_ColumnInfo;
    tcMessage     : I_TcMessage;
END_VAR
```

```
ColumnInfo[0].sName := 'colBigInt';     ColumnInfo[0].eType := E_ColumnType.BigInt;     ColumnInfo[0
].nLength := 8;     ColumnInfo[0].sProperty := 'IDENTITY(1,1)';
ColumnInfo[1].sName := 'colInteger';    ColumnInfo[1].eType := E_ColumnType.Integer;    ColumnInfo[1
].nLength := 4;
ColumnInfo[2].sName := 'colSmallInt';   ColumnInfo[2].eType := E_ColumnType.SmallInt;   ColumnInfo[2
].nLength := 2;
ColumnInfo[3].sName := 'colTinyInt';    ColumnInfo[3].eType := E_ColumnType.TinyInt;    ColumnInfo[3
].nLength := 1;
ColumnInfo[4].sName := 'colBit';        ColumnInfo[4].eType := E_ColumnType.BIT_;       ColumnInfo[4
].nLength := 1;
ColumnInfo[5].sName := 'colMoney';      ColumnInfo[5].eType := E_ColumnType.Money;      ColumnInfo[5
].nLength := 8;
ColumnInfo[6].sName := 'colFloat';      ColumnInfo[6].eType := E_ColumnType.Float;      ColumnInfo[6
].nLength := 8;
ColumnInfo[7].sName := 'colReal';       ColumnInfo[7].eType := E_ColumnType.REAL_;      ColumnInfo[7
].nLength := 4;
ColumnInfo[8].sName := 'colDateTime';   ColumnInfo[8].eType := E_ColumnType.DateTime;   ColumnInfo[8
].nLength := 4;
ColumnInfo[9].sName := 'colNText';      ColumnInfo[9].eType := E_ColumnType.NText;      ColumnInfo[9
].nLength := 256;
ColumnInfo[10].sName := 'colNChar';     ColumnInfo[10].eType := E_ColumnType.NChar;     ColumnInfo[1
0].nLength := 10;
ColumnInfo[11].sName := 'colImage';     ColumnInfo[11].eType := E_ColumnType.Image;     ColumnInfo[1
1].nLength := 256;
ColumnInfo[12].sName := 'colNVarChar';  ColumnInfo[12].eType := E_ColumnType.NVarChar;  ColumnInfo[1
2].nLength := 50;
ColumnInfo[13].sName := 'colBinary';    ColumnInfo[13].eType := E_ColumnType.Binary;    ColumnInfo[1
3].nLength := 30;
ColumnInfo[14].sName := 'colVarBinary'; ColumnInfo[14].eType := E_ColumnType.VarBinary; ColumnInfo[1
4].nLength := 20;

IF fbPLCDBCreate.Table(
    hDBID:= 1,
    sTableName:= 'myNewTable',
    pTableCfg:= ADR(ColumnInfo),
    cbTableCfg:= SIZEOF(ColumnInfo))
THEN
    IF fbPLCDBCreate.bError THEN
        TcMessage:= fbPLCDBCreate.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.2.4 FB_PLCDBReadEvt

Funktionsbaustein zum Auslesen von Datensätzen aus einer Datenbank.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBReadEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Beschreibung
nRecords	UDINT	Gibt die Anzahl der Records aus, die in Abhängigkeit von sDBSymbolName maximal abgeholt werden könnten.
eTraceLevel	TcEventSeverity [▶ 229]	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Read [▶ 180]	Lokal	Liest eine vorgegebene Anzahl an Datensätzen aus einer Datenbanktabelle mit der von Beckhoff vorgegebenen Standardtabellenstruktur.
ReadStruct [▶ 181]	Lokal	Liest eine vorgegebene Anzahl an Datensätzen aus einer Datenbanktabelle mit beliebiger Tabellenstruktur.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.2.4.1 Read

Diese Methode liest eine vorgegebene Anzahl an Datensätzen aus einer Datenbanktabelle mit der von Beckhoff vorgegebenen Standardtabellenstruktur. Die Standardtabellenstruktur wird unter anderem beim AutoLog-Modus und beim FB_DBWriteEvt-Baustein verwendet.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    sDBSymbolName: T_MaxString;
    eOrderBy: E_OrderColumn := E_OrderColumn.eColumnID;
    eOrderType: E_OrderType := E_OrderType.eOrder_ASC;
    nStartIndex: UDINT;
    nRecordCount: UDINT;
    pData: POINTER TO ST_StandardRecord;
    cbData: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
sDBSymbolName	T_MaxString	Symbolname, der in der Standardtabellenstruktur ausgelesen werden soll.
eOrderBy	E_OrderColumn.eColumnID	Sortierspalte (ID, Timestamp, Name oder Value)
eOrderType	E_OrderType.eOrder_ASC	Sortierrichtung (ASC oder DESC)
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pData	POINTER TO ST_StandardRecord	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Strukturarrays in Byte an.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBRead    : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
    ReadStruct     : ST_StandardRecord;
    tcMessage      : I_TcMessage;
END_VAR
```

```
IF fbPLCDBRead.Read(
    hDBID:= 1,
    sTableName:= 'MyTable_WithLReal',
    sDBSymbolName:= 'MyValue',
    eOrderBy:= E_OrderColumn.ID,
    eOrderType:= E_OrderType.DESC,
    nStartIndex:= 0,
```

```
    nRecordCount:= 1,
    pData:= ADR(ReadStruct),
    cbData:= SIZEOF(ReadStruct))
THEN
    IF fbPLCDBRead.bError THEN
        tcMessage := fbPLCDBRead.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der SPS:

Expression	Type	Value
ReadStruct	ST_StandardRecord	
nID	LINT	2
dtTimestamp	DATE_AND_TIME	DT#2018-2-1-16:8:8
sName	STRING(80)	'MyValue'
rValue	LREAL	15.9

6.1.1.2.4.2 ReadStruct

Diese Methode liest eine vorgegebene Anzahl an Datensätzen aus einer Datenbanktabelle mit beliebiger Tabellenstruktur.

Syntax

```
METHOD ReadStruct : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
    cbColumnNames: UDINT;
    sOrderByColumn: STRING(50);
    eOrderType: E_OrderType := E_OrderType.eOrder_ASC
    nStartIndex: UDINT;
    nRecordCount: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Adresse des Arrays, welches die zu lesenden Spaltennamen beinhaltet.
cbColumnNames	UDINT	Länge des Spaltennamen-Arrays
sOrderByColumn	STRING(50)	Name der Sortierspalte
eOrderType	E_OrderType	Sortierrichtung (ASC oder DESC)
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pData	POINTER TO BYTE	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Strukturarrays in Byte an.

Rückgabewert

Name	Typ	Beschreibung
ReadStruct	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBRead    : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
    myCustomStruct : ST_Record;
    tcMessage      : I_TcMessage;
END_VAR
```

```
TYPE ST_Record :
STRUCT
    nID        : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName      : STRING;
    nSensor1   : LREAL;
    nSensor2   : LREAL;
END_STRUCT
END_TYPE
```

```
// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';
ColumnNames[3] := 'Sensor1';
ColumnNames[4] := 'Sensor2';

IF fbPLCDBRead.ReadStruct(
    hDBID:= 1,
    sTableName:= 'MyTable_Struct',
    pColumnNames:= ADR(ColumnNames),
    cbColumnNames:= SIZEOF(ColumnNames),
    sOrderByColumn:= ColumnNames[0],
    eOrderType:= E_OrderType.DESC,
    nStartIndex:= 0,
    nRecordCount:= 1,
    pData:= ADR(myCustomStruct),
    cbData:= SIZEOF(myCustomStruct))
THEN
    IF fbPLCDBRead.bError THEN
        tcMessage:= fbPLCDBRead.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der SPS:

Expression	Type	Value
myCustomStruct	ST_Record	
nID	LINT	1
dtTimestamp	DATE_AND_TIME	DT#2018-2-1-15:17:54
sName	STRING	'MyStructVal'
nSensor1	LREAL	12.34
nSensor2	LREAL	102.5

6.1.1.2.5 FB_PLCDBWriteEvt

Funktionsbaustein zum Schreiben von Datensätzen in eine Datenbank.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBWriteEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage;
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Write [▶ 184]	Lokal	Erzeugt einen Datensatz in der von Beckhoff vorgegebenen Standardtabellenstruktur.
WriteBySymbol [▶ 185]	Lokal	Liest den Wert eines vorgegebenen ADS-Symbols aus und speichert diesen in der von Beckhoff vorgegebenen Standardtabellenstruktur ab.
WriteStruct [▶ 186]	Lokal	Erzeugt einen Datensatz mit einer beliebigen Tabellenstruktur.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.2.5.1 Write

Diese Methode erzeugt einen Datensatz in der von Beckhoff vorgegebenen Standardtabellenstruktur.

Syntax

```
METHOD Write : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pValue: POINTER TO BYTE;
    cbValue: UDINT;
    sDBSymbolName: T_MaxString;
    eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
    nRingBuffParameter: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
pValue	POINTER TO BYTE	Adresse der Variable, die in die Standardtabellenstruktur geloggt werden soll.
cbValue	UDINT	Länge der Variable, die geloggt werden soll.
sDBSymbolName	T_MaxString	Name, der in die Tabelle mit geloggt wird.
eDBWriteMode	E_WriteMode	Gibt den Schreibmodus an. (Anhängen, updaten, Ringbuffer)
nRingBuffParameter	UDINT	Zusatzparameter für den Schreibmodus "Ringbuffer".

Rückgabewert

Name	Typ	Beschreibung
Write	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Dieses Beispiel zeigt die Verwendung der Methode FB_PLCDBWriteEvt.Write:

```
VAR
    fbPLCDBWrite    : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
    myValue         : LREAL := 43.23;
    tcMessage       : I_TcMessage;
END_VAR
```

```
IF fbPLCDBWrite.Write(
    hDBID:= 1,
    sTableName:= 'myTable_WithLReal',
    pValue:= ADR(myValue),
    cbValue:= SIZEOF(myValue),
    sDBSymbolName:= 'MyValue',
    eDBWriteMode:= E_WriteMode.eADS_TO_DB_RingBuff_Count,
    nRingBuffParameter:= 3)
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der Datenbank:

ID	Timestamp	Name	Value
27	Has been dropped		
28	'2018-01-30 14:04:19'	'MyValue'	41.23
29	'2018-01-30 14:04:29'	'MyValue'	42.23
30	'2018-01-30 14:04:39'	'MyValue'	43.23

Durch die Ringbufferoption werden immer nur 3 Einträge dieses Namens in der Datenbank sein. Ältere werden gelöscht.

6.1.1.2.5.2 WriteBySymbol

Diese Methode liest den Wert eines vorgegebenen ADS-Symbols aus und speichert diesen in der von Beckhoff vorgegebenen Standardtabellenstruktur ab. Es können auch ADS-Symbole von anderen ADS-Geräten ausgelesen werden.

Syntax

```
METHOD WriteBySymbol : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    stADSDevice: ST_ADSDevice;
    stSymbol: ST_Symbol;
    eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
    nRingBuffParameter: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
stADSDevice	ST_ADSDevice	ADS-Gerät, von dem ein Symbol in die Standardtabellenstruktur geloggt werden soll.
stSymbol	ST_Symbol	Symbolname der zu schreibenden Variablen
eDBWriteMode	E_WriteMode	Gibt den Schreibmodus an. (Anhängen, updaten, Ringbuffer)
nRingBuffParameter	UDINT	Zusatzparameter für den Schreibmodus “Ringbuffer“

Rückgabewert

Name	Typ	Beschreibung
WriteBySymbol	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Dieses Beispiel zeigt die Verwendung der Methode FB_PLCDBWriteEvt.WriteBySymbol:

```
VAR
    fbPLCDBWrite    :  FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
    myValue         :  LREAL := 43.23;
    myAdsDevice     :  ST_ADSDevice;
    mySymbol        :  ST_Symbol;
    tcMessage       :  I_TcMessage;
END_VAR
```

```
// Set ADSDevice Information
myAdsDevice.sDevNetID     := '127.0.0.1.1.1';
myAdsDevice.nDevPort      := 851;
myAdsDevice.eADSRdWrtMode := E_ADSRdWrtMode.bySymbolName;
myAdsDevice.tTimeout      := T#5S;

// Set Symbol Information
mySymbol.eDataType        := E_PLCDataType.eType_LREAL;
```

```
mySymbol.sDBSymbolName    := 'MySymbol';
mySymbol.sSymbolName      := 'MAIN.myValue';
mySymbol.nBitSize         := 8;

// Call Functionblock
IF fbPLCDBWrite.WriteBySymbol(
    hDBID:= 1,
    sTableName:= 'myTable_WithLReal',
    stADSDevice:= myAdsDevice,
    stSymbol:= mySymbol,
    eDBWriteMode:= E_WriteMode.eADS_TO_DB_Append,
    nRingBuffParameter:= 1)
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der Datenbank:

ID	Timestamp	Name	Value
28	'2018-01-30 14:04:19'	'MyValue'	41.23
29	'2018-01-30 14:04:29'	'MyValue'	42.23
30	'2018-01-30 14:04:39'	'MyValue'	43.23
31	'2018-01-30 14:06:12’	‘MySymbol’	86.2

6.1.1.2.5.3 WriteStruct

Diese Methode erzeugt einen Datensatz mit einer beliebigen Tabellenstruktur.

Syntax

```
METHOD WriteStruct : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pRecord: POINTER TO BYTE;
    cbRecord: UDINT;
    pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
    cbColumnNames: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
pRecord	POINTER TO BYTE	Adresse einer Struktur, die in eine beliebige Tabellenstruktur geloggt werden soll.
cbRecord	UDINT	Länge der zu schreibenden Struktur
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Adresse des Arrays, welches die zu füllenden Spaltennamen beinhaltet.
cbColumnNames	UDINT	Länge des Spaltennamen-Arrays

Rückgabewert

Name	Typ	Beschreibung
WriteStruct	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Dieses Beispiel zeigt die Verwendung der Methode FB_PLCDBWriteEvt.WriteStruct:

```
VAR
    fbPLCDBWrite    :  FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
    myRecord        :  ST_Record;
    ColumnNames     :  ARRAY[0..4] OF STRING(50);

    systime         :  GETSYSTEMTIME;
    currentTime     :  T_FILETIME;
    tcMessage       :  I_TcMessage;
END_VAR
```

```
TYPE ST_Record :
STRUCT
    nID         : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName       : STRING;
    nSensor1    : LREAL;
    nSensor2    : LREAL;
END_STRUCT
END_TYPE
```

```
// set Values
systime(timeLoDw => currentTime.dwLowDateTime, timeHiDW => currentTime.dwHighDateTime );
myRecord.dtTimestamp := FILETIME_TO_DT(currentTime);
myRecord.sName       := 'MyStructVal';
myRecord.nSensor1    := 12.34;
myRecord.nSensor2    := 102.5;

// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';
ColumnNames[3] := 'Sensor1';
ColumnNames[4] := 'Sensor2';

// Call Functionblock
IF fbPLCDBWrite.WriteStruct(
    hDBID:= 1,
    sTableName:= 'myTable_Struct',
    pRecord:= ADR(myRecord),
    cbRecord:= SIZEOF(myRecord),
    pColumnNames:= ADR(ColumnNames) ,
    cbColumnNames:= SIZEOF(ColumnNames))
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der Datenbank:

ID	Timestamp	Name	Sensor1	Sensor2
5	'2018-01-30 15:23:26'	'MyStructVal'	12.34	102.5

6.1.1.2.6 FB_PLCDBCmdEvt

Funktionsbaustein mit zwei Methoden. Eigene SQL-Kommandos können definiert und übergeben werden. Platzhalter im SQL-Kommando können mit Strukturen in der SPS korrelieren, welche die Tabellenstruktur widerspiegeln. Der Database Server setzt die aktuellen Daten der Struktur in das SQL-Kommando ein.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBCmdEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Execute [▶ 189]	Lokal	Sendet beliebige SQL-Kommandos an die Datenbank. Zurückgelieferte Datensätze können nicht ausgelesen werden.
ExecuteDataReturn [▶ 190]	Lokal	Sendet beliebige SQL-Kommandos an die Datenbank. Eine vorgegebene Anzahl von Datensätzen kann ausgelesen werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.2.6.1 Execute

Mit dieser Methode können beliebige SQL-Kommandos an die Datenbank gesendet werden. Die Datenbankverbindung wird bei jedem Aufruf geöffnet und nach der Ausführung wieder geschlossen. Es besteht die Möglichkeit, Platzhalter im Kommando zu definieren, die vom TwinCAT Database Server vor der Ausführung mit den entsprechenden Werten befüllt werden. Zurückgelieferte Datensätze können nicht ausgelesen werden.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    hDBID: UDINT;
    pExpression: POINTER TO BYTE;
    cbExpression: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
    cbParameter: UDINT;
END_VAR
```

Beispiel

```
VAR
    fbPLCDBCmd    : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
    sCmd          : STRING (1000);
    myStruct      : ST_DataAll;
    aPara         : ARRAY[0..14] OF ST_ExpParameter;
    tcMessage     : I_TcMessage;
END_VAR
```

```
TYPE ST_DataAll :
STRUCT
    colBigInt: LINT;
    colInteger: DINT;
    colSmallInt: INT;
    colTinyInt: BYTE;
    colBit: BOOL;
    colMoney: LREAL;
    colFloat: LREAL;
    colReal: REAL;
    colDateTime: DT;
    colNText: STRING(255);
    colNChar: STRING(10);
    colImage: ARRAY[0..255] OF BYTE;
    colNVarChar: STRING(50);
    colBinary: ARRAY[0..29] OF BYTE;
    colVarBinary: ARRAY[0..19] OF BYTE;
END_STRUCT
END_TYPE
```

```
// set Parameter configuration
aPara[0].sParaName := 'colBigInt';    aPara[0].eParaType :=
E_ExpParameterType.Int64;      aPara[0].nParaSize := 8;
aPara[1].sParaName := 'colInteger';   aPara[1].eParaType :=
E_ExpParameterType.Int32;      aPara[1].nParaSize := 4;
aPara[2].sParaName := 'colSmallInt';  aPara[2].eParaType :=
E_ExpParameterType.Int16;      aPara[2].nParaSize := 2;
aPara[3].sParaName := 'colTinyInt';   aPara[3].eParaType :=
E_ExpParameterType.Byte_;      aPara[3].nParaSize := 1;
aPara[4].sParaName := 'colBit';       aPara[4].eParaType :=
E_ExpParameterType.Boolean;    aPara[4].nParaSize := 1;
aPara[5].sParaName := 'colMoney';     aPara[5].eParaType :=
E_ExpParameterType.Double64;   aPara[5].nParaSize := 8;
aPara[6].sParaName := 'colFloat';     aPara[6].eParaType :=
E_ExpParameterType.Double64;   aPara[6].nParaSize := 8;
aPara[7].sParaName := 'colReal';      aPara[7].eParaType :=
E_ExpParameterType.Float32;    aPara[7].nParaSize := 4;
aPara[8].sParaName := 'colDateTime';  aPara[8].eParaType :=
E_ExpParameterType.DateTime;   aPara[8].nParaSize := 4;
aPara[9].sParaName := 'colNText';     aPara[9].eParaType :=
E_ExpParameterType.STRING_;    aPara[9].nParaSize := 256;
aPara[10].sParaName:= 'colNChar';     aPara[10].eParaType :=
E_ExpParameterType.STRING_;   aPara[10].nParaSize := 10;
aPara[11].sParaName:= 'colImage';     aPara[11].eParaType :=
E_ExpParameterType.ByteArray; aPara[11].nParaSize := 256;
aPara[12].sParaName:= 'colNVarChar';  aPara[12].eParaType :=
E_ExpParameterType.STRING_;   aPara[12].nParaSize := 50;
aPara[13].sParaName:= 'colBinary';    aPara[13].eParaType :=
```

```
E_ExpParameterType.ByteArray; aPara[13].nParaSize := 30;
aPara[14].sParaName:= 'colVarBinary'; aPara[14].eParaType :=
E_ExpParameterType.ByteArray; aPara[14].nParaSize := 20;

// set command
sCmd := 'INSERT INTO MyTableName (colInteger, colSmallInt, colTinyInt, colBit, colMoney, colFloat,
colReal, colDateTime, colNText, colNChar, colImage, colNVarChar, colBinary, colVarBinary) VALUES
({colInteger}, {colSmallInt}, {colTinyInt}, {colBit}, {colMoney}, {colFloat}, {colReal},
{colDateTime}, {colNText}, {colNChar}, {colImage}, {colNVarChar}, {colBinary}, {colVarBinary})';

// call functionblock
IF fbPLCDBCmd.Execute(
    hDBID:= 1,
    pExpression:= ADR(sCmd),
    cbExpression:= SIZEOF(sCmd),
    pData:= ADR(myStruct),
    cbData:= SIZEOF(myStruct),
    pParameter:= ADR(aPara),
    cbParameter:= SIZEOF(aPara))
THEN
    IF fbPLCDBCmd.bError THEN
        tcMessage := fbPLCDBCmd.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.2.6.2 ExecuteDataReturn

Mit dieser Methode können beliebige SQL-Kommandos an die Datenbank gesendet werden. Die Datenbankverbindung wird bei jedem Aufruf geöffnet und nach der Ausführung wieder geschlossen. Es besteht die Möglichkeit, Platzhalter im Kommando zu definieren, die vom TwinCAT Database Server vor der Ausführung mit den entsprechenden Werten befüllt werden. Eine vorgegebene Anzahl von Datensätzen kann ausgelesen werden.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    hDBID: UDINT;
    pExpression: POINTER TO BYTE;
    cbExpression: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
    cbParameter: UDINT;
    nStartIndex: UDINT;
    nRecordCount: UDINT;
    pReturnData: POINTER TO BYTE;
    cbReturnData: UDINT;
    pRecords: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
pExpression	POINTER TO BYTE	Adresse der String-Variablen mit dem SQL Kommando
cbExpression	UDINT	Länge der String-Variablen mit dem SQL-Kommando
pData	POINTER TO BYTE	Adresse der Struktur mit den Parameterwerten
cbData	UDINT	Länge der Struktur mit den Parameterwerten
pParameter	POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter	Adresse des Strukturarrays mit den Parameterinformationen
cbParameter	UDINT	Länge des Strukturarrays mit den Parameterinformationen
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pReturnData	POINTER TO BYTE	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbReturnData	UDINT	Gibt die Größe des Strukturarrays in Byte an.
pRecords	POINTER TO BYTE	Anzahl der ausgelesenen Datensätze.

Rückgabewert

Name	Typ	Beschreibung
ExecuteDataReturn	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Parametrieren des Kommandos

Die Spaltennamen für die einzelnen Parameter werden im SQL-Kommando in geschweiften Klammern angegeben.
Beispiel: ‚SELECT * FROM MyHouse_Temperatures WHERE Room = {SelectedRoom}'. SelectedRoom muss dabei in der Struktur ST_ExpParameter entsprechend als Parametername angegeben sein.

Einige Datenbanken unterstützen das Parametrieren von SQL Klauseln nicht. (TOP/LIMIT/ROWNUM/…) Auch parametrierbare Tabellennamen werden in der Regel nicht unterstützt.

Beispiel

```
VAR
    fbPLCDBCmd       : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
    sCmd             : STRING (1000);
    stPara           : ST_ExpParameter;
    RecordAmt        : ULINT := 3;
    ReturnDataStruct : ARRAY [0..9] OF ST_DataAll;
    nRecords         : UDINT;
    tcMessage        : I_TcMessage;
END_VAR
```

```
// set Parameter configuration
stPara.eParaType := E_ExpParameterType.Int64;
stPara.nParaSize := 8;
stPara.sParaName := 'RecordAmt';

// set command with placeholder
sCmd := 'SELECT TOP ({RecordAmt}) * FROM MyTableName';

// call functionblock
IF fbPLCDBCmd.ExecuteDataReturn(
    hDBID:= 1,
    pExpression:= ADR(sCmd),
    cbExpression:= SIZEOF(sCmd),
    pData:= ADR(RecordAmt),
    cbData:= SIZEOF(RecordAmt),
```

```
    pParameter:= ADR(stPara),
    cbParameter:= SIZEOF(stPara),
    nStartIndex:= 0,
    nRecordCount:= 10,
    pReturnData:= ADR(ReturnDataStruct),
    cbReturnData:= SIZEOF(ReturnDataStruct),
    pRecords:= ADR(nRecords))
THEN
    IF fbPLCDBCmd.bError THEN
        tcMessage := fbPLCDBCmd.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.3 SQL Expert Mode

6.1.1.3.1 FB_ConfigTcDBSrvEvt

Funktionsbaustein zum Erstellen, Auslesen und Löschen von Konfigurationseinträgen für den TwinCAT Database Server.

Syntax

Definition:

```
FUNCTION_BLOCK FB_ConfigTcDBSrvEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage;
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Create [▶ 193]	Lokal	Erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server
Read [▶ 194]	Lokal	Liest die aktuelle Konfiguration des TwinCAT Database Servers
Delete [▶ 195]	Lokal	Löscht die Datenbank und die AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.3.1.1 Create

Diese Methode erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server. Optional kann ein neuer Eintrag auch nur temporär vom TwinCAT Database Server verwendet werden. Er wird dann nicht in die XML-Datei geschrieben.

Syntax

```
METHOD Create : BOOL
VAR_INPUT
    pTcDBSrvConfig: POINTER TO BYTE;
    cbTcDBSrvConfig: UDINT;
    bTemporary: BOOL := TRUE;
    pConfigID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pTcDBSrvConfig	POINTER TO BYTE	Zeiger der Konfigurationsstruktur, die erzeugt werden soll.
cbTcDBSrvConfig	UDINT	Länge der Konfigurationsstruktur
bTemporary	BOOL	Gibt an, ob die Konfiguration in die XML-Datei gespeichert werden soll.
pConfigID	POINTER TO UDINT	Rückgabezeiger der Konfigurations-ID (hDBID oder hAutoLogGrpID)

Das Anlegen von Autolog-Gruppen wird derzeit nicht unterstützt.

Rückgabewert

Name	Typ	Beschreibung
Create	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    // Any other ConfigType can be used here
    stConfigDB      : T_DBConfig_MsCompactSQL;
    tcMessage       : I_TcMessage;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Recipes.sdf';

IF fbConfigTcDBSrv.Create(
    pTcDBSrvConfig:= ADR(stConfigDB),
    cbTcDBSrvConfig:= SIZEOF(stConfigDB),
    bTemporary:= TRUE,
    pConfigID:= ADR(myConfigHandle))
THEN
    IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.3.1.2 Read

Mit dieser Methode können die aktuellen Konfigurationen des TwinCAT Database Servers gelesen werden. Wenn temporäre Konfigurationen dabei sind, werden diese entsprechend markiert.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    pDBConfig: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    cbDBConfig: UDINT;
```

```
    pAutoLogGrpConfig: POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF
ST_ConfigAutoLogGrp;
    cbAutoLogGrpConfig: UDINT;
    pDBCount: POINTER TO UDINT;
    pAutoLogGrpCount: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 249]] OF ST_ConfigDB [▶ 232]	Pointer-Adresse des Arrays, in das die Datenbankkonfigurationen geschrieben werden sollen.
cbDBConfig	UDINT	Länge des Datenbankkonfiguration-Arrays
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS [▶ 249]] OF ST_ConfigAutoLogGrp [▶ 231]	Pointer-Adresse des Arrays, in das die AutoLogGrp Konfigurationen geschrieben werden sollen.
cbAutoLogGrpConfig	UDINT	Länge des AutoLogGrp-Konfiguration-Arrays
pDBCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der Datenbankkonfigurationen gespeichert werden.
pAutoLogGrpCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der AutoLogGrp-Konfigurationen gespeichert werden.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv   : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    aDBConfig         : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    aAutoGrpConfig    : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp;
    nDbCount          : UDINT;
    nAutoGrpCount     : UDINT;
    tcMessage         : I_TcMessage;
END_VAR
```

```
IF fbConfigTcDBSrv.Read(
    pDBConfig := ADR(aDBConfig),
    cbDBConfig := SIZEOF(aDBConfig),
    pAutologGrpConfig := ADR(aAutoGrpConfig),
    cbAutoLogGrpConfig := SIZEOF(aAutoGrpConfig),
    pDBCount := ADR(nDbCount),
    pAutoLogGrpCount := ADR(nAutoGrpCount))
 THEN
    IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.3.1.3 Delete

Mit dieser Methode können Datenbank und AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers gelöscht werden.

Syntax

```
METHOD Delete : BOOL
VAR_INPUT
    eTcDBSrvConfigType: E_TcDBSrvConfigType;
    hConfigID: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
eTcDBSrvConfigType	E_TcDBSrvConfigType	Typ der zu löschenden Konfiguration (Datenbank / AutoLog-Gruppe)
hConfigID	UDINT	ID der zu löschenden Konfiguration (hDBID oder hAutoLogGrpID)

Rückgabewert

Name	Typ	Beschreibung
Delete	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    tcMessage       : I_TcMessage;
END_VAR
```

```
IF fbConfigTcDBSrv.Delete(
    eTcDBSrvConfigType := E_TcDBSrvConfigType.Database,
    hConfigID := myConfigHandle) THEN
IF fbConfigTcDBSrv.bError THEN
        tcMessage := fbConfigTcDBSrv.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.1.3.2 FB_SQLDatabaseEvt

Funktionsbaustein zum Öffnen, Schließen und Verwalten einer Datenbankverbindung.

Syntax

Definition:

```
FUNCTION_BLOCK FB_SQLDatabaseEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Connect [▶ 197]	Lokal	Öffnet eine Verbindung zu einer deklarierten Datenbank.
CreateCmd [▶ 198]	Lokal	Initialisiert eine Instanz des Bausteins FB_SQLCommandEvt [▶ 200] mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabaseEvt.
CreateSP [▶ 199]	Lokal	Initialisiert eine Instanz des Bausteins FB_SQLStoredProcedureEvt [▶ 205] mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabaseEvt Bausteins.
Disconnect [▶ 200]	Lokal	Schließt die Verbindung zur Datenbank, die von dieser Bausteininstanz geöffnet wurde.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.3.2.1 Connect

Diese Methode öffnet eine Verbindung zu einer deklarierten Datenbank.

Syntax

```
METHOD Connect : BOOL
VAR_INPUT
    hDBID: UDINT := 1;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.

Rückgabewert

Name	Typ	Beschreibung
Connect	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR
```

```
// open connection
IF fbSqlDatabase.Connect(1) THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

6.1.1.3.2.2 CreateCmd

Mit dieser Methode wird eine Instanz des Bausteins FB_SQLCommand mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabase initialisiert. Der Baustein FB_SQLCommand verwendet nur die Datenbankverbindung, die ihm über die CreateCmd-Methode zugewiesen wurde. Mehrere Instanzen vom Baustein FB_SQLCommand können mit derselben Datenbankverbindung initialisiert werden.

Die Initialisierung des Bausteins FB_SQLCommand ist im selben Zyklus abgeschlossen. Somit muss weder das Busy-Flag des Bausteins noch der Methodenrückgabewert der CreateCmd-Methode überprüft werden.

Syntax

```
METHOD CreateCmd : BOOL
VAR_INPUT
    pSQLCommand: POINTER TO FB_SQLCommandEvt;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pSQLCommand	POINTER TO FB_SQLCommand	Liefert eine neue Instanz des FB_SQLCommandEvt-Bausteins zurück.

Rückgabewert

Name	Typ	Beschreibung
CreateCmd	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR
```

```
// create a command reference
IF fbSqlDatabase.CreateCmd(ADR(fbSqlCommand)) THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
```

```
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLCommandEvt [▶ 200] zur Ausführung verwendet werden.

6.1.1.3.2.3 CreateSP

Mit dieser Methode wird eine Instanz des Bausteins FB_SQLStoredProcedureEvt mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabaseEvt initialisiert. Der Baustein FB_SQLStoredProcedureEvt verwendet nur die Datenbankverbindung, die ihm über die CreateCmd-Methode zugewiesen wurde. Mehrere Instanzen vom Baustein FB_SQLStoredProcedureEvt können mit derselben Datenbankverbindung initialisiert werden.

Die Initialisierung des Bausteins FB_SQLStoredProcedureEvt kann einige Zyklen in Anspruch nehmen. Das Busy-Flag des Bausteins oder der Methodenrückgabewert der CreateCmd-Methode muss überprüft werden bevor der Baustein einsatzbereit ist.

Syntax

```
METHOD CreateSP : BOOL
VAR_INPUT
    sProcedureName: T_MaxString;
    pParameterInfo: POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter;
    cbParameterInfo: UDINT;
    pSQLProcedure: POINTER TO FB_SQLStoredProcedureEvt;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sProcedureName	T_MaxString	Gibt den Namen der Prozedur an, welche ausgeführt werden soll.
pParameterInfo	POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter	Pointer-Adresse zu der Parameter-Infoliste.
cbParameterInfo	UDINT	Gibt die Länge der Parameter-Infoliste an.
pSQLProcedure	POINTER TO FB_SQLStoredProcedureEvt	Liefert eine neue Instanz des Bausteins FB_SQLStoredProcedureEvt zurück.

Rückgabewert

Name	Typ	Beschreibung
CreateSP	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase   : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
    ParaInfo        : ST_SQLSPParameter;
END_VAR
```

```
ParaInfo.sParameterName     := '@Customer_ID';
ParaInfo.eParameterType     := E_SPParameterType.Input;
ParaInfo.eParameterDataType := E_ColumnType.BigInt;
ParaInfo.nParameterSize     := 8;

IF fbSQLDatabase.CreateSP('dbo.SP_GetCustomerPositions', ADR(ParaInfo), SIZEOF(ParaInfo), ADR(fbSQLS
toredProcedure)) THEN
    IF fbSQLDatabase.bError THEN
        nState:=255;
    ELSE
        nState:= nState+1;
    END_IF
END_IF
```

Darauffolgend kann der FB_SQLStoredProcedureEvt [▶ 205] verwendet werden, um die gespeicherte Prozedur auszuführen.

6.1.1.3.2.4 Disconnect

Diese Methode schließt die Verbindung zur Datenbank, welche von dieser Bausteininstanz geöffnet wurde.

Syntax

```
METHOD Disconnect : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Disconnect	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR
```

```
// disconnect from database
IF fbSqlDatabase.Disconnect() THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

6.1.1.3.3 FB_SQLCommandEvt

Funktionsbaustein zum Ausführen von SQL-Kommandos. Zuvor muss er mit dem Baustein FB_SQLDatabaseEvt initialisiert werden.

Syntax

Definition:

```
FUNCTION_BLOCK FB_SQLCommandEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Execute [▶ 201]	Lokal	Sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabaseEvt [▶ 196] geöffnete Datenbankverbindung an die Datenbank.
ExecuteDataReturn [▶ 202]	Lokal	Sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabaseEvt [▶ 196] geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom Baustein FB_SQLResultEvt [▶ 203] übergeben werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.3.3.1 Execute

Diese Methode sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pSQLCmd	POINTER TO BYTE	Gibt die Pointer-Adresse einer String-Variablen mit dem auszuführenden SQL-Befehl an.
cbSQLCmd	UDINT	Gibt die Länge des SQL-Befehls an, der ausgeführt werden soll.

Rückgabewert

Name	Typ	Beschreibung
Execute	POINTER TO BYTE	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Nutzt das vom FB_SQLDatabaseEvt.CreateCmd() [▶ 196] erstellte Kommando.

```
VAR
    fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
    tcMessage    : I_TcMessage;
END_VAR
```

```
// you can generate this with the SQL Query Editor
sCmd := 'INSERT INTO myTable_Double ( Timestamp, Name, Value) VALUES ( $'2018-01-31 14:59:27$', $'Te
mperature$', 21.3)';

// call sql command
IF fbSQLCommand.Execute(ADR(sCmd), SIZEOF(sCmd)) THEN
    IF fbSQLCommand.bError THEN
        tcMessage := fbSQLCommand.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

6.1.1.3.3.2 ExecuteDataReturn

Diese Methode sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom Baustein FB_SQLResult übergeben werden.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
    pSQLDBResult: POINTER TO FB_SQLResult;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pSQLCmd	POINTER TO BYTE	Gibt die Pointer-Adresse einer String-Variablen mit dem auszuführenden SQL-Befehl an.
cbSQLCmd	UDINT	Gibt die Länge des SQL-Befehls an, der ausgeführt werden soll.
pSQLDBResult	POINTER TO FB_SQLResult [▶ 203]	Gibt eine neue Instanz des Bausteins FB_SQLResult zurück.

Rückgabewert

Name	Typ	Beschreibung
ExecuteDataReturn	POINTER TO BYTE	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Nutzt das vom FB_SQLDatabaseEvt.CreateCmd() [▶ 196] erstellte Kommando.

```
VAR
    fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
    tcMessage    : I_TcMessage;
END_VAR
```

```
// you can generate this with the SQL Query Editor
sCmd := 'SELECT ID, Timestamp, Name, Value FROM myTable_Double';

// call sql command
IF fbSQLCommand.ExecuteDataReturn(ADR(sCmd), SIZEOF(sCmd), ADR(fbSqlResult)) THEN
    IF fbSQLCommand.bError THEN
        nState := 255;
    ELSE
         tcMessage := fbSQLCommand.ipTcResult;
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLResultEvt [▶ 203] genutzt werden, um die Daten auszulesen.

6.1.1.3.4 FB_SQLResultEvt

Der Baustein dient zum Auslesen der gepufferten Datensätze.

Syntax

Definition:

```
FUNCTION_BLOCK FB_SQLResultEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Read [▶ 204]	Lokal	Liest eine vorgegebene Anzahl von Datensätzen aus den im TwinCAT Database Server gepufferten Ergebnisdaten aus.
Release [▶ 205]	Lokal	Gibt vom TwinCAT Database Server gepufferte Daten wieder frei.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.3.4.1 Read

Diese Methode liest eine vorgegebene Anzahl von Datensätzen aus den im TwinCAT Database Server gepufferten Ergebnisdaten aus.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    nStartIndex: UDINT := 0;
    nRecordCount: UDINT := 1;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    bWithVerifying: BOOL := FALSE;
    bDataRelease: BOOL := TRUE;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pData	POINTER TO BYTE	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Strukturarrays in Byte an.
bWithVerifying	BOOL	Rückgabedaten werden mit pData Strukturarray verglichen und ggf. angepasst.
bDataRelease	BOOL	Gibt die gepufferten Daten wieder frei.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlResult : FB_SQLResultEvt(sNetID:='', tTimeout := T#5S);
    aReadStruct : ARRAY[1..5] OF ST_StandardRecord;
END_VAR
```

```
// get values from internal tc db srv storage
IF fbSqlResult.Read(2, 3, ADR(aReadStruct), SIZEOF(aReadStruct), FALSE, TRUE) THEN
    IF fbSqlResult.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Ergebnis in der SPS:

Expression	Type	Value
aReadStruct	ARRAY [1..5] OF ST...	
aReadStruct[1]	ST_StandardRecord	
nID	LINT	9
dtTimestamp	DATE_AND_TIME	DT#2018-1-31-15:4:59
sName	STRING(80)	'Temperature'
rValue	LREAL	21.3
aReadStruct[2]	ST_StandardRecord	
nID	LINT	10
dtTimestamp	DATE_AND_TIME	DT#2018-1-31-15:5:59
sName	STRING(80)	'Temperature'
rValue	LREAL	21.2

6.1.1.3.4.2 Release

Mit dieser Methode können vom TwinCAT Database Server gepufferte Daten wieder freigegeben werden.

Syntax

```
METHOD Release : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Release	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

6.1.1.3.5 FB_SQLStoredProcedureEvt

Funktionsbaustein zum Ausführen von gespeicherten Prozeduren (Stored Procedures) der Datenbank. Zuvor muss dieser mit dem Baustein FB_SQLDatabaseEvt initialisiert werden.

Syntax

Definition:

```
FUNCTION_BLOCK FB_SQLStoredProcedureEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
```

```
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
Execute [▶ 207]	Lokal	Sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom Baustein FB_SQLDatabaseEvt [▶ 196] geöffnete Datenbankverbindung an die Datenbank.
ExecuteDataReturn [▶ 207]	Lokal	Sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom FB_SQLDatabaseEvt [▶ 196]-Baustein geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom FB_SQLResultEvt [▶ 203]-Baustein übergeben werden.
Release [▶ 208]	Lokal	Gibt die Parameterinformationen der gespeicherten Prozedur (Stored Procedure), die bei der Initialisierung übergeben wurden, wieder frei.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.3.5.1 Execute

Diese Methode sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom Baustein FB_SQLDatabaseEvt geöffnete Datenbankverbindung an die Datenbank.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pParameterStrc	POINTER TO BYTE	Pointer-Adresse zur Parameterstruktur, die der Prozedur übergeben wird.
cbParameterStrc	UDINT	Länge der Parameterstruktur.

Rückgabewert

Name	Typ	Beschreibung
Execute	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Nutzt die zuvor mit FB_SQLDatabaseEvt.CreateSP() [▶ 196] erstellte Stored Procedure

```
VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID:='', tTimeout := T#5S);
    Customer_ID          : LINT;
    tcMessage            : I_TcMessage;
END_VAR
```

```
IF fbSQLStoredProcedure.Execute(pParameterStrc := ADR(Customer_ID) , cbParameterStrc:= SIZEOF(Custom
er_ID)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

6.1.1.3.5.2 ExecuteDataReturn

Diese Methode sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom FB_SQLDatabase-Baustein geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom FB_SQLResult-Baustein übergeben werden.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
    pSQLDBResult: POINTER TO FB_SQLDBResultEvt;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pParameterStrc	POINTER TO BYTE	Pointer-Adresse zur Parameterstruktur, die der Prozedur übergeben wird.
cbParameterStrc	UDINT	Länge der Parameterstruktur
pSQLDBResult	POINTER TO FB_SQLDBResultEvt	Gibt eine neue Instanz des Bausteins FB_SQLDBResultEvt zurück.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Nutzt die zuvor mit FB_SQLDatabaseEvt.CreateSP() [▶ 196] erstellte Stored Procedure

```
VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID:='', tTimeout := T#5S);
    Customer_ID          : LINT;
    tcMessage            : I_TcMessage;
END_VAR
```

```
IF fbSQLStoredProcedure.ExecuteDataReturn(pParameterStrc := ADR(Customer_ID), cbParameterStrc:= SIZE
OF(Customer_ID), pSQLDBResult := ADR(fbSqlResult)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLResultEvt [▶ 203] genutzt werden, um die Daten auszulesen.

6.1.1.3.5.3 Release

Diese Methode gibt die Parameterinformationen der gespeicherten Prozedur (Stored Procedure), die bei der Initialisierung übergeben wurden, wieder frei.

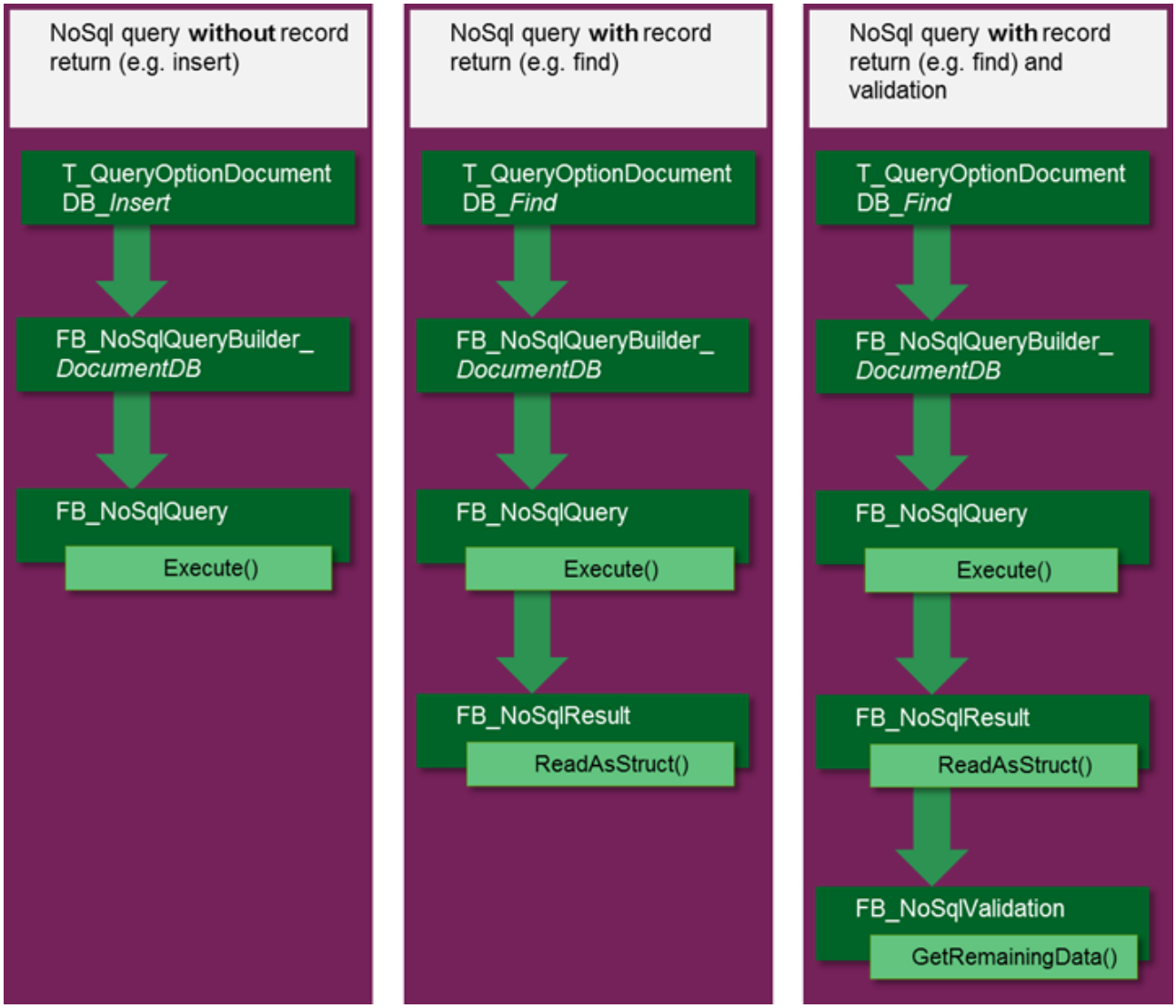
Syntax

```
METHOD Release : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Release	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

6.1.1.4 NoSQL Expert Mode



6.1.1.4.1 Query Builder

Um möglichst alle NoSQL-Datenbanken zu unterstützen, gibt es Abfrage-Bausteine, welche unterschiedliche Parametrierungen für Abfragen anbieten. Diese Bausteine werden dann als Interface den Methoden des FB_NoSQLQuery übergeben.

6.1.1.4.1.1 FB_NoSQLQueryBuilder_DocumentDB

Funktionsbaustein zum Definieren einer Abfrage für die Datenbank. Die Abfrage wird mit dem FB_NoSQLQueryEvt [▶ 212] abgeschickt. Das Aufrufen der Methode Build ist nicht notwendig.

Syntax

Definition:

```
FUNCTION_BLOCK FB_NoSQLQueryBuilder_DocumentDB
VAR_INPUT
    eQueryType : E_DocumentDbQueryType;
    sCollectionName : T_MAXSTRING;
    pQueryOptions: POINTER TO BYTE;
    cbQueryOptions : UDINT;
END_VAR
VAR_OUTPUT
END_VAR
```

Eingänge

Name	Type	Beschreibung
eQueryType	E_DocumentDbQueryType [▶ 237]	Typ der Abfrage, die zur Datenbank geschickt wird.
sCollectionName	T_MAXSTRING	Name der Collection, welche Ziel der Abfrage ist.
pQueryOptions	POINTER TO BYTE	Gibt die Adresse zu den Query Optionen [▶ 238] an.
cbQueryOptionsr	UDINT	Länge der Query Optionen.

Methoden

Name	Definitionsort	Beschreibung
Build [▶ 210]	Lokal	[optional] Diese Methode generiert aus den eingestellten Parametern eine Abfrage für den FB_NoSQLQueryEvt [▶ 212].

Beispiel:

```
VAR
    fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
    sFilter : T_MAXSTRING;
    stOptions : T_QueryOptionDocumentDB_Find;
END_VAR
```

```
// Set your settings before you run the query
stOptions.pFilter:= ADR(sFilter);
stOptions.cbFilter:= SIZEOF(sFilter);

fbNoSQLQueryBuilder_DocumentDB.eQueryType:=E_DocumentDbQueryType.Find;
fbNoSQLQueryBuilder_DocumentDB.sCollectionName:= 'MyCollectionName';
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions:= ADR(stOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions:= SIZEOF(stOptions);
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.4.1.1.1 Build

Diese Methode wird im Falle eines FB_NoSQLQuery [▶ 212]Evt (entweder mit Execute oder ExecuteDataReturn) automatisch vor dem Absenden des Queries aufgerufen. Sie erzeugt eine TwinCAT 3 Database Server spezifischen Query aus den angegebenen Parametern des QueryBuilders.

6.1.1.4.1.2 FB_NoSQLQueryBuilder_TimeSeriesDB

Funktionsbaustein zum Definieren einer Abfrage einer TimeSeries Datenbank. Die Abfrage wird mit dem FB_NoSQLQueryEvt [▶ 212] abgeschickt. Das Aufrufen der Methode Build ist nicht notwendig. Datenstrukturen [▶ 362] können dabei mit Attributen beschrieben werden, um individuelle Einstellungen zu treffen.

Eingänge

Name	Type	Beschreibung
pQueryOptions	POINTER TO BYTE	Gibt die Adresse zu den Query Optionen [▶ 240] an.
cbQueryOptions	UDINT	Länge der Query Optionen.

Syntax

Definition:

```
FUNCTION_BLOCK FB_NoSQLQueryBuilder_TimeSeriesDB
VAR_INPUT
    pQueryOptions : POINTER TO BYTE;
    cbQueryOptions : UDINT;
END_VAR
VAR_OUTPUT
END_VAR
```

Methoden

Name	Definitionsort	Beschreibung
Build [▶ 210]	Lokal	[optional] Diese Methode generiert aus den eingestellten Parametern eine Abfrage für den FB_NoSQLQueryEvt [▶ 212].

Bespiel:

```
VAR
    fbNoSQLQueryBuilder_TimeSeriesDB : FB_NoSQLQueryBuilder_TimeSeriesDB;
    QueryOption_TSDB_Insert : T_QueryOptionTimeSeriesDB_Insert;
    fbNoSqlQueryEvt : FB_NoSQLQueryEvt(sNetID := '', tTimeout := T#15S);
    MyStructArray: ARRAY[1..1000] OF MyStruct;
END_VAR
```

```
CASE nState OF
    1: // init
        fbNoSQLQueryBuilder_TimeSeriesDB.pQueryOptions := ADR(QueryOption_TSDB_Insert);
        fbNoSQLQueryBuilder_TimeSeriesDB.cbQueryOptions := SIZEOF(QueryOption_TSDB_Insert);

        QueryOption_TSDB_Insert.sTableName := 'MeasurementName';
        QueryOption_TSDB_Insert.sDataType := 'MyStruct';
        QueryOption_TSDB_Insert.pSymbol := ADR(MyStructArray);
        QueryOption_TSDB_Insert.cbSymbol := SIZEOF(MyStructArray);
        QueryOption_TSDB_Insert.nDataCount := 1000; // ArrayLength
        QueryOption_TSDB_Insert.nStartTimestamp := F_GetSystemTime(); // get current twincat time
        QueryOption_TSDB_Insert.nCycleTime := 1000; // equivalent to 1 ms

    2: // write values
        IF fbNoSqlQueryEvt.Execute(dbid, fbNoSQLQueryBuilder_TimeSeriesDB) THEN
            IF fbNoSqlQueryEvt.bError THEN
                // do some error handling here
            ELSE
                // success
            END_IF
        END_IF
END_CASE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.4.1.2.1 Build

Diese Methode wird im Falle eines FB_NoSQLQuery [▶ 212]Evt (entweder mit Execute oder ExecuteDataReturn) automatisch vor dem Absenden des Queries aufgerufen. Sie erzeugt eine TwinCAT 3 Database Server spezifischen Query aus den angegebenen Parametern des QueryBuilders.

6.1.1.4.2 FB_NoSQLQueryEvt

Funktionsbaustein zu Ausführen von Anfragen an eine NoSQL-Datenbank. Als Eingangsparameter wird bei den Methoden ein QueryBuilder-Funktionsbaustein verwendet, welcher die Abfrage beschreibt.

Syntax

Definition:

```
FUNCTION_BLOCK FB_NoSQLQueryEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Methoden

Name	Definitionsort	Beschreibung
Execute [▶ 213]	Lokal	Sendet die durch den QueryBuilder Baustein eingestellte Abfrage an die Datenbank.
ExecuteDataReturn [▶ 214]	Lokal	Sendet die durch den QueryBuilder Baustein eingestellte Abfrage an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom Baustein FB_NoSqlResultEvt [▶ 215] übergeben werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.4.2.1 Execute

Diese Methode sendet eine Abfrage an die NoSQL-Datenbank, welche vorher mit dem I_NoSQLQueryBuilder [▶ 209]-Funktionsbaustein eingestellt wurde.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    hDBID: UDINT;
    iNoSQLQueryBuilder: I_NoSQLQueryBuilder;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	ID der eingestellten Datenbankkonfiguration
iNoSSQLQuery Builder	I_NoSQLQueryBuilder	Vorparametrierter QueryBuilder-Baustein. Je nach Datenbank ist dieser unterschiedlich.

Rückgabewert

Name	Typ	Beschreibung
Execute	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Nutzt den QueryBuilder, um die entsprechende Abfrage auszuführen.

```
VAR
    fbNoSQLQuery : FB_NoSQLQueryEvt(sNetID := '', tTimeout := T#5S);
    fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
    InsertQueryOptions: T_QueryOptionDocumentDB_Insert;
    myDBID : UDINT := 1;
    sDocument : STRING(1000);
    TcMessage : I_TcMessage;
END_VAR
```

```
// set QueryInputs
fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.InsertOne;
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions := ADR(InsertQueryOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(InsertQueryOptions);

// set insert parameter:
sDocument := '{Name : "MyValue", Value : 123.456}';
InsertQueryOptions.pDocuments:= ADR(sDocument);
InsertQueryOptions.cbDocuments:= SIZEOF(sDocument);

// call nosql command
IF fbNoSQLQuery.Execute(myDBID, fbNoSQLQueryBuilder_DocumentDB) THEN
```

```
    IF fbNoSQLQuery .bError THEN
        TcMessage := fbNoSQLQuery.ipTcResult
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Zuvor wird der FB_NoSQLQueryEvt über den FB_NoSQLQueryBuilder_DocumentDB [▶ 209] parametriert. Je nach Querytyp gibt es verschiedene Optionen, wie die des T_QueryOptionDocumentDB_Insert [▶ 239], um das einzufügende Dokument zu setzen.

6.1.1.4.2.2 ExecuteDataReturn

Diese Methode führt eine Abfrage an eine NoSql-Datenbank aus, welche zuvor über den I_NoSqlQueryBuilder-Funktionsbaustein eingestellt wurde. Dabei wird die übergebene Instanz des Typs FB_NoSqlResultEvt mit Rückgabewerten gefüllt.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    hDBID : UDINT;
    iNoSSQLQueryBuilder: I_NoSQLQueryBuilder;
    pNoSQLResult: POINTER TO FB_NoSQLResultEvt;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	ID der eingestellten Datenbankkonfiguration
iNoSQLQueryBuilder	I_NoSQLQueryBuilder	Vorparametrierter QueryBuilder-Funktionsbaustein, welcher die abzuschickende Abfrage definiert.
pNoSQLResult	POINTER TO FB_NoSSQLResultEvt	Gibt die Adresse zum FB_NoSQLResultEvt an, mit dem die Ergebnisse ausgelesen werden können.

Rückgabewert

Name	Typ	Beschreibung
ExecuteDataReturn	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.
nDataCount	UDINT	[optional] Anzahl der zurückgelieferten Datensätze

Nutzt den QueryBuilder, um die entsprechende Abfrage auszuführen.

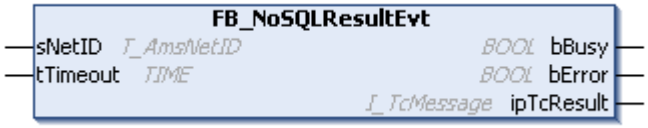
```
VAR
    fbNoSqlQuery : FB_NoSQLQueryEvt(sNetID := '', tTimeout := T#5S);
    fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB
    FindQueryOptions : T_QueryOptionDocumentDB_Find;
    fbNoSqlResult : FB_NoSQLResultEvt(sNetID := '', tTimeout := T#5S);
    myDBID : UDINT := 1;
    sFilter : STRING(255);
    sSort: STRING(255);
    sProjection: STRING(255);
    TcMessage : I_TcMessage;
END_VAR
```

```
// set QueryInputs:
fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.Find;
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions := ADR(FindQueryOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(FindQueryOptions);

//set Find Parameter ([optional] sort, projection):
sFilter := '{}'; // read all data from database
FindQueryOptions.pFilter:= ADR(sFilter);
FindQueryOptions.cbFilter:= SIZEOF(sFilter);

// call nosql query:
IF fbNoSqlQuery.ExecuteDataReturn(myDBID, fbNoSqlQuery, ADR(fbNoSqlResult)) THEN
```

```
    IF fbNoSqlQuery.bError THEN
        TcMessage := fbNoSqlQuery.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Zuvor wird der FB_NoSQLQueryEvt über den FB_NoSQLQueryBuilder_DocumentDB [▶ 209] parametriert. Je nach Querytyp gibt es verschiedene Optionen, wie die des T_QueryOptionDocumentDB_Find [▶ 238], um den Filter, die Sortierung oder die Projektion zu definieren.

6.1.1.4.3 FB_NoSQLResultEvt



Funktionsbaustein zum Auslesen gepufferter Datensätze.

Die Datensätze müssen zuvor mit dem FB_NoSQLQueryEvt [▶ 212] beim Aufruf der ExecuteDataReturn [▶ 214]-Methode aus der Datenbank abgerufen werden. Dabei wird der FB_NoSQLResultEvt zur Initialisierung angegeben. Diese können dann entweder als Struktur der SPS oder als Zeichenfolge ausgelesen werden.

Syntax

Definition:

```
FUNCTION BLOCK FB_SQLResultEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.
nDataCount	UDINT	Gibt die Anzahl der zurückgelieferten verfügbaren Datensätze vom Aufruf des FB_NoSQLQueryEvt.ExecuteDataReturn() [▶ 214] an.

Methoden

Name	Definitionsort	Beschreibung
ReadAsString [▶ 216]	Lokal	Liest eine vorgegebene Anzahl von Datensätzen aus den im TwinCAT Database Server gepufferten Ergebnisdaten als Json-String aus.
ReadAsStruct [▶ 217]	Lokal	Liest eine vorgegebene Anzahl von Datensätzen aus den im TwinCAT Database Server gepufferten Ergebnisdaten in die angegebene Struktur.
Release [▶ 218]	Lokal	Gibt vom TwinCAT Database Server gepufferte Daten wieder frei.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.4.3.1 ReadAsString

Diese Methode liest eine vorgegebene Anzahl an Datensätzen aus dem im TwinCAT Database Server gepufferten Ergebnisdaten aus. Dabei wird ein Array von Strings angegeben, in das diese Daten als JSON kopiert werden sollen.

Syntax

```
METHOD ReadAsString : BOOL
VAR_INPUT
    nStartIndex: UDINT := 0;
    nRecordCount: UDINT := 1;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    nMaxDocumentSize : UDINT;
    bDataRelease: BOOL := TRUE;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätze an.
pData	POINTER TO BYTE	Adresse des Stringarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Stringarrays in Byte an.
nMaxDocumentSize	UDINT	Gibt die Maximalgröße eines einzelnen Json-Dokuments von pData an.
bDataRelease	BOOL	Gibt die gepufferten Daten wieder frei.

Rückgabewert

Name	Typ	Beschreibung
ReadAsString	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel:

```
VAR
    fbNoSqlResult : FB_NoSQLResultEvt(sNetID := '', tTimeout := T#5S);
    aRead_Json : ARRAY[0..2] OF STRING(1000);
    TcMessage : I_TcMessage;
END_VAR
```

```
IF fbNoSqlResult.ReadAsString(
    nStartIndex:= 0,
    nRecordCount:= 3,
    pData:= ADR(aRead_Json),
    cbData:= SIZEOF(aRead_Json),
    MaxDocumentSize:= SIZEOF(aRead_Json[0]),
    bDataRelease:= TRUE)
THEN
    IF fbNoSqlResult.bError THEN
        TcMessage := fbNoSqlResult.ipTcResult;
        nstate := 255;
    ELSE
        nstate := nstate+1;
    END_IF
END_IF
```

6.1.1.4.3.2 ReadAsStruct

Diese Methode liest eine vorgegebene Anzahl an Datensätzen aus den gepufferten Ergebnisdaten aus. Dabei wird eine Struktur oder ein Array einer Struktur angegeben, in das die Daten geschrieben werden sollen. Das Datentypschema dieser Struktur sollte dabei möglichst dem der ausgelesenen Daten entsprechen. Dabei werden die Variablennamen und die des Datensatzes verglichen. Mit Hilfe einer Validierung gibt es die Möglichkeit, Abweichungen zu erkennen und darauf zu reagieren.

Falls die Anforderung besteht, unterschiedliche Namen in der Datenbank und in der SPS zu verwenden, können in der Struktur die Namen mit dem Attribut *‚ElementName‘* mit dem zugeordneten Namen aus der Datenbank beschrieben werden.

Syntax

```
METHOD ReadAsStruct: BOOL
VAR_INPUT
    nStartIndex: UDINT := 0;
    nRecordCount: UDINT := 1;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    bValidate: BOOL := FALSE;
    pNoSQLValidation : POINTER TO FB_NoSQLValidationEvt;
    bDataRelease: BOOL := TRUE;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pData	POINTER TO BYTE	Adresse des Strukturarray, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Strukturarray in Byte an.
bValidate	BOOL	Rückgabedaten werden mit pData Strukturarray verglichen und ggf. angepasst.
pNoSQLValidation	POINTER TO FB_NoSQLValidationEvt	Adresse des FB_NoSQLValidationEvt-Bausteins, der weitere Informationen zur Validierung des Aufrufs zur Verfügung stellt.
bDataRelease	BOOL	Gibt die gepufferten Daten wieder frei.

Rückgabewert

Name	Typ	Beschreibung
ReadAsStruct	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel:

```
VAR
    fbNoSQLResult: FB_NoSQLResultEvt(sNetID := '', tTimeout := T#5S);
    aRead : ARRAY[0..2] OF ST_MyDataStruct;
    fbNoSQLValidation : FB_NoSQLValidationEvt(sNetID := '', tTimeout := #5S);
END_VAR
```

```
IF fbNoSQLResult.ReadAsStruct(
    nStartIndex:= 0,
    nRecordCount:= 3,
    pData:= ADR(aRead),
    cbData:= SIZEOF(aRead),
    bValidate:= TRUE,
    pNoSQLValidation:= ADR(fbNoSQLValidation),
    bDataRelease:= TRUE)
THEN
    IF fbNoSQLResult.bError THEN
        TcMessage := fbNoSQLResult.ipTcResult;
        nstate := 255;
    ELSE
        nstate := nstate+1;
    END_IF
END_IF
```

6.1.1.4.3.3 Release

Mit dieser Methode können vom TwinCAT Database Server gepufferte Daten wieder freigegeben werden.

Syntax

```
METHOD Release : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Release	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

6.1.1.4.4 FB_NoSQLValidationEvt

Funktionsbaustein zum Auslesen der Validierungsereignisse und –Ergebnisse, die beim Lesen der Daten mit dem FB_NoSQLResultEvt [▶ 215] aufgetreten sind. Die Initialisierung dieses Bausteins erfolgt über die CreateValidation Methode des NoSQLResult und bezieht sich auf den letzten Aufruf der ReadAsStruct [▶ 217]-Methode.

Syntax

Definition:

```
FUNCTION_BLOCK FB_NoSQLValidationEvt
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResult: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 228]	Nachrichten Interface vom TwinCAT 3 Eventlogger, welches Details zum Rückgabewert zur Verfügung stellt.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Methoden

Name	Definitionsort	Beschreibung
GetIssues [▶ 220]	Lokal	Liest eine Auflistung der Validierungsereignisse als Stringarray.
GetRemainingData [▶ 221]	Lokal	Liest die Daten als String aus, welche keinem Element in der Struktur in der PLC zugeordnet werden konnten.
Release [▶ 221]	Lokal	Gibt die gepufferten Daten im TwinCAT Database Server frei.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.4.4.1 GetIssues

Liest die aufgetretenen Validierungsereignisse in ein Array vom Typ T_MAXSTRING. Diese enthalten Informationen über übrig gebliebene oder nicht zugeordneten Datensätze oder welche Elemente der PLC-Struktur nicht befüllt wurden.

Syntax

```
METHOD GetIssues : BOOL
VAR_INPUT
    pData : POINTER TO BYTE;
    cbData: UDINT;
    bDataRelease : BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pData	POINTER TO BYTE	Adresse des Arrays vom Type T_MAXSTRING, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Stringarrays in Byte an.
bDataRelease	BOOL	Gibt die gepufferten Daten wieder frei.

Rückgabewert

Name	Typ	Beschreibung
ReadAsString	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel:

```
VAR
    fbNoSqlValidation : FB_NoSQLValidation(sNetID := '', tTimeout := t#15S);
    aIssues : ARRAY[0..1000] OF T_MAXSTRING;
END_VAR
```

```
IF fbNoSqlValidation.GetIssues(
    pData:= ADR(aIssues),
    cbData:= SIZEOF(aIssues),
    bDataRelease:= TRUE)
THEN
    IF fbNoSqlValidation.bError THEN
        TcMessage := fbNoSqlValidation.ipTcResult;
        nstate := 255;
    ELSE
        nstate := nstate+1;
    END_IF
END_IF
```

6.1.1.4.4.2 GetRemainingData

Mit dieser Methode können nach der Validierung die übrig gebliebenen Daten in Json-Format ausgelesen werden. Zum Beispiel von Datensätzen, welche der PLC-Struktur nicht zugehordnet werden konnten.

Syntax

```
METHOD GetRemainingData : BOOL
VAR_INPUT
    pData : POINTER TO BYTE;
    cbData : UDINT;
    cbDocument : UDINT;
    bDataRelease : BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pData	POINTER TO BYTE	Adresse des Stringarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Stringarrays in Byte an.
cbDocument	UDINT	Gibt die Stringlänge des Strings im Array an.
bDataRelease	BOOL	Gibt die gepufferten Daten wieder frei.

Rückgabewert

Name	Typ	Beschreibung
GetRemainingData	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel:

```
VAR CONSTANT
    cDocumentSize : UDINT := 1000;
END_VAR
VAR
    fbNoSqlValidation : FB_NoSQLValidation(sNetID := '', tTimeout := t#15S);
    aRemainingData : ARRAY[0..1000] OF STRING(cDocumentSize);
END_VAR
```

```
IF fbNoSqlValidation.GetRemainingData(
    pData:= ADR(aRemainingData),
    cbData:= SIZEOF(aRemainingData),
    cbDocument:= cDocumentSize,
    bDataRelease:= TRUE)
THEN
    IF fbNoSqlValidation.bError THEN
        TcMessage := fbNoSqlValidation.ipTcResult;
        nstate := 255;
    ELSE
        nstate := nstate+1;
    END_IF
END_IF
```

6.1.1.4.4.3 Release

Gibt die Validierungsergebnisse im Speicher frei.

Syntax

```
METHOD Release : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Release	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

6.1.1.4.5 Helper

Diese Funktionsbausteine bieten zum Beispiel im Umgang mit Datentypen hilfreiche Funktionen.

6.1.1.4.5.1 FB_NoSQLObjectId_MongoDB

Der Funktionsbaustein zum Parsen der ObjectId von der MongoDB. In der SPS wird sie durch den Datentypen T_ObjectId_MongoDB [▶ 237] beschrieben.

Syntax

```
FUNCTION_BLOCK FB_NoSQLObjectId_MongoDB
VAR_INPUT
    ObjectId : T_ObjectId_MongoDB;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
ObjectId	T_ObjectId_MongoDB	12-Byte großer Datentyp zur Beschreibung der ObjectId.

Eigenschaften

Name	Typ	Beschreibung
eTraceLevel	TcEventSeverity	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.
nId	UDINT	Nicht eindeutige, fortlaufende Nummer
nMachineId	UDINT	Identifikation der Maschine
nProcessId	UINT	Identifikation des Schreibprozesses
tTimestamp	DATE_AND_TIME	Zeitstempel des Datensatz

Methoden

Name	Definitionsort	Rückgabewert	Beschreibung
ToString	Lokal	STRING(36)	Gibt die Id als String mit Typbezeichnung zurück. Bsp.: ‚ObjectId(„5be15c11afa6ec72b107dafaf“)‘
ValueOf	Lokal	STRING(24)	Gibt nur die Id als String zurück. Bsp.: ‚5be15c11afa6ec72b107dafaf‘

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.1.5 Unterstützung des Tc3_Eventlogger

Der TwinCAT 3 Database Server unterstützt den TwinCAT 3 Eventlogger (TwinCAT 3 Version 4022.20). Dieser ermöglicht es über ein Interface, Details zu Ereignissen der Bausteine auszulesen. Weitere Informationen zum Eventlogger finden Sie in den TwinCAT 3 Basis Bibliotheken.

Alle Funktionsbausteine des TwinCAT 3 Database Servers unterstützen das Interface des Tc3 Eventloggers. Dazu wird als Rückgabewert der Bausteine das Interface Tc3_Eventlogger.I_TcMessage [▶ 228] verwendet. Zusätzlich zum Rückgabewert, steht die Eigenschaft *eTraceLevel* zur Verfügung, um die Ereignisgewichtung zu bestimmen.

Eigenschaften

Name	Typ	Zugriff	Beschreibung
eTraceLevel	TcEventSeverity [▶ 229]	Get, Set	Gibt die Gewichtung der Ereignisse an. Nur Ereignisse über diesem Wert werden an das TwinCAT System gesendet.

Beispiel:

Bei einem exemplarischen Funktionsbaustein FB_PLCDBWrite geben wir folgende Gewichtung an:

```
fbPLCDBWriteEvt.eTraceLevel := TcEventSeverity.Warning;
```

Hier werden nun alle Ereignisse gesendet, welche mindestens eine Warnung darstellen. Ereignisse der Gewichtung „Information" werden in diesem Fall ignoriert.

Die Tc3_Database Funktionsbausteine selbst haben als Ausgang *ipTcResult* vom Datentyp Tc3_Eventlogger.I_TcMessage. Alle angebotenen Funktionen dieses Interfaces können genutzt werden.

In diesem Beispiel wird zunächst der Baustein aufgerufen.

```
1:  // Call Functionblock
    IF fbPLCDBWriteEvt.WriteStruct(
        hDBID:= 1,
        sTableName:= 'myTable_Struct',
        pRecord:= ADR(myRecord),
        cbRecord:= SIZEOF(myRecord),
        pColumnNames:= ADR(ColumnNames) ,
        cbColumnNames:= SIZEOF(ColumnNames))
    THEN
        IF fbPLCDBWriteEvt.bError THEN
            myTcMessage := fbPLCDBWriteEvt.ipTcResult
            nState := 255;
        ELSE
            nState := 0;
        END_IF
    END_IF
```

Im Fehlerfall möchten wir nun den Eventtext in der Laufzeitumgebung anfordern. Hierzu kann die Methode *RequestEventText* verwendet werden. Mit der *nLangId =1031* wird der Fehlercode in deutscher Sprache ausgelesen. Dies ist eine der vielen Funktionen der Schnittstelle Schnittstelle Tc3_Eventlogger.I_TcMessage [▶ 228].

```
255://Request EventText
    IF myTcMessage.RequestEventText(1031,
        ADR(MyEventString),
        SIZEOF(MyEventString))THEN
        nState := 0;
    END_IF
```

6.1.1.5.1 I_TcEventBase

In dieser Basisschnittstelle sind Methoden und Eigenschaften eines Ereignisses definiert.

Methoden

Name	Beschreibung
EqualsTo [▶ 224]	Vergleicht das Ereignis mit einer anderen Instanz.
EqualsToEventClass [▶ 225]	Vergleicht die Ereignisklasse des Ereignisses mit einer anderen Ereignisklasse.
EqualsToEventEntryEx [▶ 226]	Vergleicht die Ereignisdefinition des Ereignisses mit einer anderen Ereignisdefinition.
GetJsonAttribute [▶ 226]	Liefert das Json-Attribut.
RequestEventClassName [▶ 227]	Fragt den Namen der Ereignisklasse an.
RequestEventText [▶ 227]	Liefert den Text zum Ereignis.

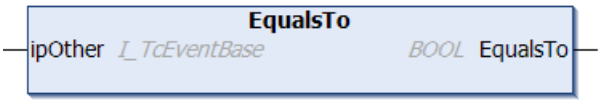
Eigenschaften

Name	Typ	Zugriff	Beschreibung
eSeverity	TcEventSeverity [▶ 229]	Get	Liefert die Severity.
EventClass	GUID	Get	Liefert die GUID der Ereignisklasse.
ipSourceInfo	I_TcSourceInfo	Get	Liefert einen Zeiger auf die Quelldefinition.
nEventId	UDINT	Get	Liefert die ID des Ereignisses.
stEventEntry	TcEventEntry	Get	Liefert die Ereignisdefinition.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.20	PC oder CX (x64, x86, ARM)	Tc3_EventLogger

6.1.1.5.1.1 EqualsTo



Diese Methode führt einen Vergleich mit einem am Eingang angegebenen anderen Ereignis aus.

Syntax

```
METHOD EqualsTo : BOOL
VAR_INPUT
    ipOther : I_TcEventBase;
END_VAR
```

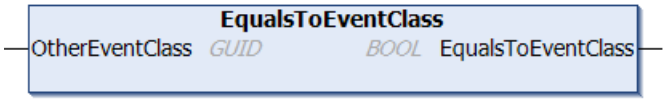
Eingänge

Name	Typ	Beschreibung
ipOther	I_TcEventBase	Zu vergleichendes Ereignis

Rückgabewert

Name	Typ	Beschreibung
EqualsTo	BOOL	Liefert TRUE, wenn die Ereignisse übereinstimmen.

6.1.1.5.1.2 EqualsToEventClass



Diese Methode führt einen Vergleich mit einer am Eingang angegebenen anderen Ereignisklasse aus.

Syntax

```
METHOD EqualsToEventClass : BOOL
VAR_INPUT
    OtherEventClass : GUID
END_VAR
```

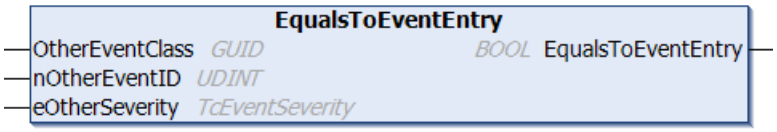
Eingänge

Name	Typ	Beschreibung
OtherEventClass	GUID	Zu vergleichende Ereignisklasse.

Rückgabewert

Name	Typ	Beschreibung
EqualsToEventClass	BOOL	Liefert TRUE, wenn die Ereignisklassen übereinstimmen.

6.1.1.5.1.3 EqualsToEventEntry



Diese Methode führt einen Vergleich mit einem am Eingang angegebenen anderen Ereignis aus.

Syntax

```
METHOD EqualsToEventEntry : BOOL
VAR_INPUT
    OtherEventClass : GUID;
    nOtherEventID   : UDINT;
    eOtherSeverity  : TcEventSeverity;
END_VAR
```

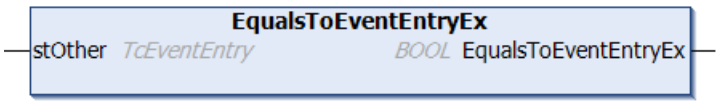
Eingänge

Name	Typ	Beschreibung
OtherEventClass	GUID	Ereignisklasse des zu vergleichenden Ereignisses.
nOtherEventID	UDINT	Event-ID des zu vergleichenden Ereignisses.
eOtherSeverity	TcEventSeverity	Event-Severity des zu vergleichenden Ereignisses.

Rückgabewert

Name	Typ	Beschreibung
EqualsToEventEntry	BOOL	Liefert TRUE, wenn die Ereignisse übereinstimmen.

6.1.1.5.1.4 EqualsToEventEntryEx



Diese Methode führt einen Vergleich mit einem am Eingang angegebenen anderen Ereignis aus.

Syntax

```
METHOD EqualsToEventEntryEx : BOOL
VAR_INPUT
    stOther : TcEventEntry;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
stOther	TcEventEntry	Zu vergleichendes Ereignis.

Rückgabewert

Name	Typ	Beschreibung
EqualsToEventEntryEx	BOOL	Liefert TRUE, wenn die Ereignisse übereinstimmen.

6.1.1.5.1.5 GetJsonAttribute

Diese Methode liefert das Json-Attribut.

Syntax

```
METHOD GetJsonAttribute : HRESULT
VAR_INPUT
    sJsonAttribute : REFERENCE TO STRING;
    nJsonAttribute : UDINT;
END_VAR
```

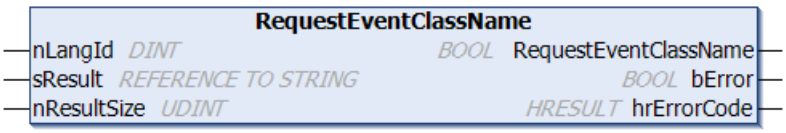
Eingänge

Name	Typ	Beschreibung
sJsonAttribute	REFERENCE TO STRING	Referenz auf eine Variable vom Typ String
nJsonAttribute	UDINT	Länge der String-Variable

Rückgabewert

Name	Typ	Beschreibung
GetJsonAttribute	HRESULT	Liefert S_OK, wenn der Methodenaufruf erfolgreich war. Liefert ERROR_BAD_LENGTH, wenn die Länge der Variable zu klein ist. Andernfalls wird HRESULT als Fehlercode zurückgegeben.

6.1.1.5.1.6 RequestEventClassName



Diese Methode liefert den Namen der Ereignisklasse.

Syntax

```
METHOD RequestEventClassName : BOOL
VAR_INPUT
    nLangId     : DINT;
    sResult     : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
nLangId	DINT	Spezifiziert die Sprach-ID Englisch (en-US) = 1033 Deutsch (de-DE) = 1031 …
sResult	REFERENCE TO STRING	Referenz auf eine Variable vom Typ String
nResultSize	UDINT	Größe der String-Variable in Bytes

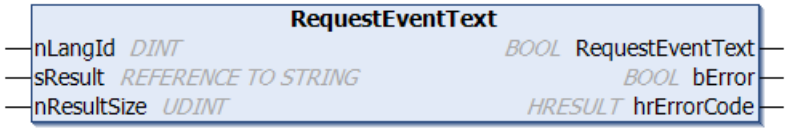
Rückgabewert

Name	Typ	Beschreibung
RequestEventClassName	BOOL	Liefert TRUE, sobald die Abfrage beendet wurde. Wenn die asynchrone Abfrage noch aktiv ist, liefert der Rückgabewert FALSE. Die Methode muss so lange aufgerufen werden, bis der Rückgabewert TRUE ist.

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Liefert FALSE, wenn der Methodenaufruf erfolgreich war. Liefert TRUE, wenn ein Fehler aufgetreten ist.
hrErrorCode	HRESULT	Liefert S_OK, wenn der Methodenaufruf erfolgreich war. Im Falle eines Fehlers wird ein Fehlercode ausgegeben.

6.1.1.5.1.7 RequestEventText



Diese Methode liefert den Ereignistext.

Syntax

```
METHOD RequestEventText : BOOL
VAR_INPUT
    nLangId     : DINT;
```

```
    sResult     : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
nLangId	DINT	Spezifiziert die Sprach-ID Englisch (en-US) = 1033 Deutsch (de-DE) = 1031 …
sResult	REFERENCE TO STRING	Referenz auf eine Variable vom Typ String
nResultSize	UDINT	Größe der String-Variable in Bytes

Rückgabewert

Name	Typ	Beschreibung
RequestEventText	BOOL	Liefert TRUE, sobald die Abfrage beendet wurde. Wenn die asynchrone Abfrage noch aktiv ist, liefert der Rückgabewert FALSE. Die Methode muss so lange aufgerufen werden, bis der Rückgabewert TRUE ist.

Ausgänge

Name	Typ	Beschreibung
bError	BOOL	Liefert FALSE, wenn der Methodenaufruf erfolgreich war. Liefert TRUE, wenn ein Fehler aufgetreten ist.
hrErrorCode	HRESULT	Liefert S_OK, wenn der Methodenaufruf erfolgreich war. Im Falle eines Fehlers wird ein Fehlercode ausgegeben.

6.1.1.5.2 I_TcMessage

Diese Schnittstelle stellt Methoden und Eigenschaften für das Nachrichten-Handling bereit.

Vererbungshierarchie

I_TcEventBase [▶ 223]

I_TcMessage

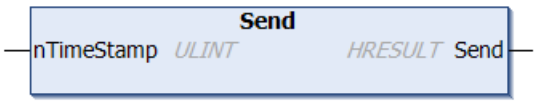
Methoden

Name	Beschreibung
Send [▶ 229]	Sendet eine Nachricht

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS-Bibliotheken
TwinCAT v3.1.4022.20	PC oder CX (x64, x86, ARM)	Tc3_EventLogger

6.1.1.5.2.1 Send



Diese Methode sendet die Nachricht.

Syntax

```
METHOD Send : HRESULT
VAR_INPUT
    nTimeStamp: ULINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
nTimeStamp	ULINT	0: Aktueller Zeitstempel wird verwendet > 0: Externer Zeitstempel in 100 Nanosekunden seit dem 1. Januar 1601 (UTC).

Rückgabewert

Name	Typ	Beschreibung
Send	FB_ HRESULT	Liefert S_OK, wenn der Methodenaufruf erfolgreich war, andernfalls ein HRESULT als Fehlercode

6.1.1.5.3 Datentypen

6.1.1.5.3.1 TcEventSeverity

Definiert die Severity des Ereignisses.

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE TcEventSeverity : (
    Verbose  := 0,
    Info     := 1,
    Warning  := 2,
    Error    := 3,
    Critical := 4);
END_TYPE
```

6.1.2 Datentypen

6.1.2.1 Config

6.1.2.1.1 E_DatabaseType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_DatabaseType :
(
    MS_Compact_SQL := 0,
```

```
    MS_Access := 1,
    MS_SQL := 2,
    ASCII := 3,
    ODBC_MySQL := 4,
    ODBC_PostgreSQL := 5,
    ODBC_Oracle := 6,
    ODBC_DB2 := 7,
    ODBC_InterBase := 8,
    ODBC_Firebird := 9,
    XML := 10,
    OCI_Oracle := 11,
    NET_MySQL := 12,
    AzureSQL := 13,
    MS_Excel := 14,
    AS400ISeries := 15,
    OleDB_Database := 16,
    Odbc_Database := 17,
    SQLite:=18,
    ODP_Oracle := 19
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.2 E_DBAuthentication

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_DBAuthentication:
(
    None:= 0,
    UserNamePassword := 1,
    x509Cert := 2,
    GSSAPI := 3,
    LDAP := 4
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.3 E_OdbcSubType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_OdbcSubType:
(
    Unknown:= 0,
    MySQL := 1,
    Oracle := 2,
    Postgre := 3,
    DB2 := 4
    Firebird := 5
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.4 E_TcDBSrvConfigType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_TcDBSrvConfigType :
(
    Database := 0,
    AutoLogGroup := 1,
    DBSrvSettings := 2,
    Symbol := 3,
    ADSDevice := 4,
    Table := 5,
    SymbolList := 6
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.5 ST_ConfigAutoLogGrp

Diese Struktur wird für die Methode Read des Bausteins FB_ConfigTcDBSrvEvt [▶ 160] verwendet. Alle konfigurierten AutoLog-Gruppen werden in einem Array dieser Struktur in die SPS gelesen.

Syntax

Definition:

```
TYPE ST_ConfigAutoLogGrp :
STRUCT
    sName: T_MaxString;
    hAutoLogGrpID: UDINT;
    hDBID: UDINT;
    sTableName: T_MaxString;
    stADSDev: ST_ADSDevice;
    eWriteMode: E_WriteMode;
    nCycleTime: TIME;
    nSymbolCount: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sName	T_MaxString	Gruppenname
hAutoLogGrpID	UDINT	ID der deklarierten AutoLog-Gruppe
hDBID	UDINT	ID der zugeordneten Datenbank
sTableName	T_MaxString	Tabellenname
stADSDev	ST_ADSDevice [▶ 246]	ADS-Geräte-Informationen
eWriteMode	E_WriteMode [▶ 245]	Schreibmodus
nCycleTime	TIME	Zykluszeit
nSymbolCount	UDINT	Anzahl der Symbole

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.6 ST_ConfigDB

Diese Struktur wird für die Methode Read des Bausteins FB_ConfigTcDBSrvEvt [▶ 160] verwendet. Alle konfigurierten Datenbankverbindungen werden in einem Array dieser Struktur in die SPS gelesen.

Syntax

Definition:

```
TYPE ST_ConfigDB :
STRUCT
    sName: T_MaxString;
    hDBID: DINT;
    eDBType: E_DatabaseType;
    sServer: STRING(80);
    sDatabase: STRING(80);
    bTemp: BOOL;
END_STRUCT
END_TYPE
```

Name	Typ	Parameter
sName	T_MaxString	Verbindungsname
hDBID	DINT	ID der deklarierten Datenbank
eDBType	E_DatabaseType [▶ 229]	Datenbanktyp
sServer	STRING (80)	Servername
sDatabase	STRING (80)	Datenbankname
bTemp	BOOL	TRUE, wenn die Verbindung nur temporär gespeichert wurde.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.7 ST_ConnStringParameter

Syntax

Definition:

```
TYPE ST_ConnStringParameter
STRUCT
    sName: T_MaxString;
    sValue: T_MaxString;
END_STRUCT
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8 ConfigType

Für die unterstützten Datenbanktypen sind unterschiedliche Datenbankkonfigurationsstrukturen vorgesehen.

6.1.2.1.8.1 T_DBConfig_ASCII

Beschreibt die Datenbankkonfigurationsstruktur für die ASCII-Datei.

Syntax

Definition:

```
TYPE T_DBConfig_ASCII
STRUCT
    sServer: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sServer	T_MaxString	Dateipfad zur ASCII-Datei

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8.2 T_DBConfig_MSAccess

Beschreibt die Datenbankkonfigurationsstruktur für eine Microsoft Access Datenbank.

Syntax

Definition:

```
TYPE T_DBConfig_MSAccess
STRUCT
    sServer: T_MaxString;
    sProvider: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sServer	T_MaxString	Dateipfad zur Access Datenbank
sProvider	T_MaxString	Access 2000 – Access 2003: „Microsoft.Jet.OLEDB.4.0“ Access 2007: „Microsoft.ACE.OLEDB.12.0“

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8.3 T_DBConfig_MsCompactSQL

Beschreibt die Datenbankkonfigurationsstruktur für eine Microsoft Compact SQL Datenbank.

Syntax

Definition:

```
TYPE T_DBConfig_MsCompactSQL
STRUCT
    sServer: T_MaxString;
    sPassword: T_MaxString;
```

```
    bAuthentification: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sServer	T_MaxString	Dateipfad zur Microsoft Compact SQL (*.sdf)
sPassword	T_MaxString	Passwort für die Datenbank
bAuthentification	BOOL	TRUE, falls die Datenbank durch ein Passwort geschützt wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8.4 T_DBConfig_MsExcel

Beschreibt die Datenbankkonfigurationsstruktur für eine Microsoft-Excel-Datei.

Syntax

Definition:

```
TYPE T_DBConfig_MsExcel
STRUCT
    sServer: T_MaxString;
    sProvider: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sServer	T_MaxString	Dateipfad zur Microsoft-Excel-Datei
sProvider	T_MaxString	„Microsoft.Jet.OLEDB.4.0“ oder „Microsoft.ACE.OLEDB.12.0“

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8.5 T_DBConfig_MsSQL

Beschreibt die Datenbankkonfigurationsstruktur für eine Microsoft SQL Datenbank.

Syntax

Definition:

```
TYPE T_DBConfig_MsSQL
STRUCT
    sServer: T_MaxString;
    sProvider: T_MaxString;
    sDatabase: T_MaxString;
    sUserID: T_MaxString;
    sPassword: T_MaxString;
    bAuthentification: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sServer	T_MaxString	Geben Sie hier den Namen ihres SQL-Servers an. Beispiel: "TESTSERVER\SQLEXPRESS"
sProvider	T_MaxString	"SQLOLEDB" oder der Provider des SQL Native Clients z.B. "SQLNCLI10"
Database	T_MaxString	Tragen Sie hier den gewünschten Namen der Datenbank ein.
sUserID	T_MaxString	Geben Sie hier den Benutzernamen an.
sPassword	T_MaxString	Geben Sie hier das zum Benutzer passende Passwort an.
bAuthentification	BOOL	Setzen Sie diese Variable auf TRUE, falls die durch UserId und Passwort angegebene Authentifizierung stattfinden soll.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8.6 T_DBConfig_Odbc

Beschreibt die Datenbankkonfigurationsstruktur für eine Datenbank mit Odbc-Schnittstelle.

Syntax

Definition:

```
TYPE T_DBConfig_NET_MySQL
STRUCT
    eOdbcSubType: E_OdbcSubType
    nParameterCount: UINT;
    arrParameter: ARRAY [0..MAX_CONFIGPARAMETER] OF ST_ConnStringParameter;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
eOdbcSubType	E_OdbcSubType [▶ 230]	Beschreibt eine im vollen Funktionsumfang unterstützte Odbc-Datenbank [▶ 230].
nParameterCount	UINT	Anzahl der Parameter für den Connectition-Strings
arrParameter	ARRAY [0..MAX_CONFIGPARAMETER] OF ST_ConnStringParameter [▶ 232];	Array von Parametern für den Connection-String vom Typ ST_ConnStringParameter [▶ 232].

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8.7 T_DBConfig_SQLite

Beschreibt die Datenbankkonfigurationsstruktur für eine SQLite.

Syntax

Definition:

```
TYPE T_DBConfig_SQLite
STRUCT
    sServer: T_MaxString;
    sPassword: T_MaxString;
    bAuthentification: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sServer	T_MaxString	Dateipfad zur SQLite
sPassword	T_MaxString	Passwort für die Datenbank
bAuthentification	BOOL	TRUE, falls die Datenbank durch ein Passwort geschützt wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.1.8.8 T_DBConfig_XML

Beschreibt die Datenbankkonfigurationsstruktur für eine XML-Datei im angepassten Datenbankformat.

Syntax

Definition:

```
TYPE T_DBConfig_XML
STRUCT
    sServer: T_MaxString;
    sSchema: T_MaxString;
    sDatabase: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sServer	T_MaxString	Name und Pfad der XML-Datei
sSchema	T_MaxString	Name und Pfad der XSD-Datei
sDatabase	T_MaxString	Beschreibt den Namen der Datenbank. Die XML-, XSD- und XSL-Datei werden für diesen Datenbanktyp automatisch erzeugt.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.2 NoSQL

6.1.2.2.1 E_NoSQLDatabaseType

Syntax

```
{attribute 'qualified_only'}
TYPE E_NoSQLDatabaseType :
(
    UnknownType := 0,
    DocumentDB := 1,
    Binary := 2,
```

```
    TimeSeriesDB := 3
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.2.2 E_DocumentDBQueryType

Syntax

```
{attribute 'qualified_only'}
TYPE E_DocumentDbQueryType :
(
    InsertOne := 1,
    InsertMany := 2,
    UpdateOne := 3,
    UpdateMany := 4 ,
    Find := 5,
    Aggregation := 6,
    Delete := 7,
    DeleteMany := 8,
    CreateCollection := 9,
    DropCollection := 10
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.2.3 T_ObjectId_MongoDB

Ein Datentyp, welcher den Datenbank-spezifischen Datentyp „ObjectId“ abbildet. Dieser besteht aus 12 Bytes mit unterschiedlicher Bedeutung:

Tab. 1: ObjectId auf Byteebene

1	2	3	4	5	6	7	8	9	10	11	12
Timestamp				MachineId			ProcessId		Id		

Um die einzelnen Elemente herauszuparsen, steht der FB_NoSQLObjectId_MongoDB [▶ 222] zur Verfügung.

Syntax

```
TYPE T_ObjectId_MongoDB : ARRAY[0..11] OF BYTE;
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.2.4 QueryOptions

6.1.2.2.4.1 DocumentDB

6.1.2.2.4.1.1 T_QueryOptionDocumentDB_Find

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Find
STRUCT
    pFilter: POINTER TO BYTE;
    cbFilter: UDINT;
    pSort: POINTER TO BYTE;
    cbSort: UDINT;
    pProjection: POINTER TO BYTE;
    cbProjection: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
pFilter	POINTER TO BYTE	Gibt die Adresse zum Suchfilter an, nach welchen die Collection durchsucht werden soll.
cbFilter	UDINT	Länge des Suchfilters.
pSort	POINTER TO BYTE	Gibt die Adresse zur Sortierung an, nach welcher die Collection sortiert werden soll.
cbSort	UDINT	Länge der Sortierung
pProjection	POINTER TO BYTE	Gibt die Adresse der Darstellung an, wie die Daten der Collection dargestellt werden sollen.
cbProjection	UDINT	Länge der Darstellung.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.2.2.4.1.2 T_QueryOptionDocumentDB_Aggregate

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Aggregate
STRUCT
    pPipeStages: POINTER TO BYTE;
    cbPipeStages: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
pPipeStages	POINTER TO BYTE	Gibt die Adresse zum Stagedokument an. In diesem können mehrere Stages definiert werden.
cbPipeStages	UDINT	Länge des Stagedokuments.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.2.2.4.1.3 T_QueryOptionDocumentDB_Insert

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Insert
STRUCT
    pDocuments: POINTER TO BYTE;
    cbDocuments: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
pDocuments	POINTER TO BYTE	Gibt die Adresse zu einem oder mehreren Dokumenten an, die zu einer Collection hinzugefügt werden soll.
cbDocuments	UDINT	Länge des Dokuments.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.2.2.4.1.4 T_QueryOptionDocumentDB_Update

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Update
STRUCT
    pFilter: POINTER TO BYTE;
    cbFilter: UDINT;
    pDocuments: POINTER TO BYTE;
    cbDocuments: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
pFilter	POINTER TO BYTE	Gibt die Adresse zum Suchfilter an, nach welchen die Collection durchsucht werden soll.
cbFilter	UDINT	Länge des Suchfilters.
pDocuments	POINTER TO BYTE	Gibt die Adresse der Dokumente an, deren Werte in die Collection übernommen werden sollen.
cbDocuments	UDINT	Länge der Dokumente

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.2.2.4.1.5 T_QueryOptionDocumentDB_Delete

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Delete
STRUCT
    pFilter: POINTER TO BYTE;
    cbFilter: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
pFilter	POINTER TO BYTE	Gibt die Adresse zum Suchfilter an, nach welchen die Collection durchsucht werden soll.
cbFilter	UDINT	Länge des Suchfilters.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.2.2.4.2 TimeSeriesDB

6.1.2.2.4.2.1 T_QueryOptionTimeSeriesDB_Insert

Diese Struktur dient zum Schreiben von Daten in eine Zeitreihendatenbank. Als Symbol kann ein Array einer Struktur angegeben werden. Jedes einzelne Array-Element wird als Reihe mit einem eindeutigen Zeitstempel in der Datenbank abgebildet. Die Zeitstempel der einzelnen Reihen werden unter Angabe eines Anfangszeitstempels und des Zeitzyklus zwischen den Datensätzen generiert.

Syntax

Definition:

```
TYPE T_QueryOptionTimeSeriesDB_Insert
STRUCT
    pSymbol         : PVOID;
    cbSymbol        : UDINT;
    sDataType       : STRING;
    nDataCount      : UDINT := 1;
    sTableName      : T_MaxString;
    nCycleTime      : UDINT := 100000;
    nStartTimestamp : ULINT := 0;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
pSymbol	POINTER TO BYTE	Zeiger auf das Symbol, welches in die Datenbank geschrieben werden soll
cbSymbol	UDINT	Länge des angegebenen Symbolzeigers
sDataType	STRING	(einfache) Typbezeichnung des angegebenen Symbols
nDataCount	UDINT	Anzahl der Datensätze im übergebenen Array-Symbol
sTableName	T_MaxString	Name der Tabelle oder Tabellen-Äquivalents
nCycleTime	UDINT	Zeitlicher Abstand der zyklischen Datensätze
nStartTimestamp	ULINT	Zeitstempel des ersten Datensatzes in TwinCAT Zeit: Anzahl der 100ns-Intervalle seit dem 1.Januar 1601. Weiterführend: Auslesen der aktuellen TwinCAT Zeit

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.2.2.4.2.2 T_QueryOptionTimeSeriesDB_Query

Diese Struktur dient zum Lesen von Daten aus einer Zeitreihendatenbank. Die Abfrage wird als String übergeben und kann anschließend an den FB_NoSQLQueryBuilder_TimeSeriesDB [▶ 210] übergeben werden.

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Find
STRUCT
    pQuery: POINTER TO BYTE;
    cbQuery: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
pQuery	POINTER TO BYTE	Gibt den Zeiger auf die Abfrage(String) an
cbQuery	UDINT	Gibt die Länge des Zeigers der Abfrage an

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4022.20	PC oder CX (x86)	Tc3_Database

6.1.2.2.5 E_TimeSeriesDbQueryType

Syntax

```
{attribute 'qualified_only'}
TYPE E_ TimeSeriesDbQueryType:
(
    Insert := 1,
    Query := 2
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.3 SQL

6.1.2.3.1 E_ColumnType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_ColumnType :
(
    BigInt := 0,
    Integer := 1,
```

```
    SmallInt := 2,
    TinyInt := 3,
    BIT_ := 4,
    Money := 5,
    Float := 6,
    REAL_ := 7,
    DateTime := 8,
    NText := 9,
    NChar := 10,
    Image := 11,
    NVarChar := 12,
    Binary := 13,
    VarBinary := 14
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.3.2 E_SPParameterType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_SPParameterType :
(
    Input := 0,
    Output := 1,
    InputOutput := 2,
    ReturnValue := 3,
    OracleCursor := 4
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.3.3 E_VerifyResult

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_VerifyResult:
(
    check_None:= 0,
    check_OK := 1,
    check_Error:= 2,
    check_TypeWarning:= 3,
    check_TypeError := 4,
    check_DataLengthWarning := 5,
    check_DataLengthError := 6
);
END_TYPE
```

Werte

Name	Beschreibung
check_None	SPS-Struktur ist nicht verifiziert.
check_OK	Keine Unterschiede zwischen SPS-Struktur und Datenbank-Tabellenstruktur
check_TypeWarning	Unterschiedliche Datentypen bezüglich der Vorzeichen zwischen SPS- und Datenbank- Datentypen z. B. UINT <> INT
check_TypeError	Unterschiedliche Datentypen bezüglich Vorzeichen zwischen SPS-Struktur und Datenbank-Tabellenstruktur
check_DataLengthWarning	Unterschiedliche Länge zwischen SPS-Struktur und Datenbank-Tabellenstruktur. Der Datensatz kann trotzdem abgebildet werden, jedoch können Daten verloren gehen.
check_DataLengthError	Unterschiedliche Länge zwischen SPS-Struktur und Datenbank-Tabellenstruktur. Der Datensatz kann nicht abgebildet werden.

6.1.2.3.4 ST_SQLSPParameter

Diese Struktur wird für den Baustein FB_SQLStoredProcedureEvt [▶ 205] benötigt, für die Beschreibung der verschiedenen Parameter der auszuführenden Prozedur.

Syntax

Definition:

```
TYPE ST_SQLSPParameter :
STRUCT
    eParameterType: E_SPParameterType;
    eParameterDataType: E_ColumnType;
    nParameterSize: UDINT;
    sParameterName: STRING(50);
END_STRUCT
END_TYPE
```

Name	Typ	Beschreibung
eParameterType	E_SPParameterType [▶ 242]	Parametertyp (INPUT, OUTPUT ...)
eParameterDataType	E_ColumnType [▶ 241]	Parametertyp
nParameterSize	UDINT	Parameterlänge
sParameterName	STRING (50)	Parametername

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4 PLC

6.1.2.4.1 E_ADSRdWrtMode

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_ADSRdWrtMode :
(
    bySymbolName := 1,
    IGroup_IOffset := 2
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.2 E_ErrorType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_ErrorType :
(
    noError := 0,
    InternalError,
    DataBaseError,
    ADSError
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.3 E_ExpParameterType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_ExpParameterType :
(
    NULL := 0,
    Boolean := 1,
    Byte_ := 2,
    Int16 := 3,
    Int32 := 4,
    Int64 := 5,
    UInt16 := 6,
    UInt32 := 7,
    UInt64 := 8,
    DateTime := 9,
    Float32 := 10,
    Double64 := 11,
    STRING_ := 12,
    ByteArray := 13,
    Struct_ := 14,
    XMLTAGName := 15
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.4 E_OrderColumn

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_OrderColumn :
(
```

```
    ID := 0,
    Timestamp := 1,
    Name := 2,
    Value := 3
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.5 E_OrderType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_OrderType :
(
    ASC := 0,
    DESC := 1
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.6 E_PLCDataType

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_PLCDataType :
(
    eType_BOOL := 0,
    eType_BYTE := 1,
    eType_SINT := 2,
    eType_INT := 3,
    eType_DINT := 4,
    eType_UINT := 5,
    eType_UDINT := 6,
    eType_WORD := 7,
    eType_DWORD := 8,
    eType_LREAL := 9,
    eType_REAL := 10,
    eType_LINT := 11,
    eType_ULINT := 12,
    eType_BigType := 13
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.7 E_WriteMode

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_WriteMode :
(
    eADS_TO_DB_Update := 0,
    eADS_TO_DB_Append := 1,
    eADS_TO_DB_RingBuff_Time := 2,
    eADS_TO_DB_RingBuff_Count := 3
);
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.8 ST_ADSDevice

Beschreibt das ADS Device, welches für Methoden des FB_PLCDBWriteEvt [▶ 182]-Bausteins mit angegeben werden muss.

Syntax

Definition:

```
TYPE ST_ADSDevice :
STRUCT
    sDevNetID: T_AmsNetId;
    nDevPort: T_AmsPort;
    eADSRdWrtMode: E_ADSRdWrtMode;
    tTimeout: TIME;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sDevNetID	T_AmsNetId	NetID des ADS-Geräts
nDevPort	T_AmsPort	AMS Port
eADSRdWrtMode	E_ADSRdWrtMode [▶ 243]	Verbindungsmodus IGroup_IOffset / bySymbol
tTimeout	TIME	ADS-Verbindungs-Timeout

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.9 ST_AutoLogGrpStatus

Liefert Informationen zur jeweiligen AutoLog-Gruppe.

Syntax

Definition:

```
TYPE ST_AutoLogGrpStatus :
STRUCT
    hAutoLogGrpID: UDINT;
    nCycleCount: UDINT;
    hrErrorCode: HRESULT;
    eErrorType: E_ErrorType;
    bError: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Parameter

Name	Typ	Beschreibung
hAutoLogGrpID	UDINT	ID der deklarierten AutoLog-Gruppe
nCycleCount	UDINT	Anzahl der bisher durchgeführten Zyklen
hrErrorCode	HRESULT	HRESULT-Fehlercode
eErrorType	E_ErrorType [▶ 244]	Fehlertyp
bError	BOOL	TRUE, wenn ein Fehler aufgetreten ist.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.10 ST_ColumnInfo

Syntax

Definition:

```
TYPE ST_ColumnInfo :
STRUCT
    sName: STRING(50);
    sProperty: STRING;
    nLength: UDINT;
    eType: E_ColumnType;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sName	STRING (50)	Name der Spalte
sProperty	STRING	String für zusätzliche Spalteneigenschaften
nLength	UDINT	Maximallänge (bei Strings und Bytestreams)
eType	E_ColumnType [▶ 241]	Spaltentyp

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.11 ST_ExpParameter

Diese Struktur wird für den Baustein FB_PLCCmd [▶ 187] benötigt, um die Beschreibung der verschiedenen Parameter (Platzhalter) in dem SQL-Kommando bereitzustellen.

Syntax

Definition:

```
TYPE ST_ExpParameter:
STRUCT
    sParaName : T_MaxString;
    nParaSize : UDINT;
```

```
    eParaType : E_ExpParameterType;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sParaName	T_MaxString	Name des Parameters (Platzhalter)
nParaSize	UDINT	Länge des Parameterwertes
eParaType	E_ExpParameterType [▶ 244]	Datentyp des Parameters

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.12 ST_StandardRecord

Wenn Sie mit der Standard-Tabellenstruktur des TwinCAT Database Servers arbeiten möchten, können Sie diese Struktur in der SPS verwenden.

Diese Struktur kann nicht im Zusammenhang mit Microsoft Access Datenbanken verwendet werden, da dieser Datenbanktyp keinen 64-Bit-Integer-Datentyp unterstützt. In diesem Fall sollten Sie die Struktur ST_StandardRecord_MSAccess [▶ 248] verwenden.

Syntax

Definition:

```
TYPE ST_StandardRecord :
STRUCT
    nID: LINT;
    dtTimestamp: DT;
    sName: STRING(80);
    rValue: LREAL;
END_STRUCT
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.13 ST_StandardRecord_MSAccess

Wenn Sie mit der Standard-Tabellenstruktur des TwinCAT Database Servers arbeiten möchten, können Sie diese Struktur in der SPS verwenden. Diese Struktur ist speziell für Microsoft Access Datenbanken vorgesehen, da dieser Datenbanktyp keinen 64-Bit-Integer-Datentyp unterstützt.

Syntax

Definition:

```
TYPE ST_StandardRecord_MSAccess:
STRUCT
    nID: DINT;
    dtTimestamp: DT;
    sName: STRING(80);
    rValue: LREAL;
END_STRUCT
END_TYPE
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.2.4.14 ST_Symbol

Beschreibt das ADS-Symbol, welches für Methoden des Bausteins FB_PLCDBWriteEvt [▶ 182] mit angegeben werden muss.

Syntax

Definition:

```
TYPE ST_Symbol :
STRUCT
    sSymbolName: T_MaxString;
    sDBSymbolName: T_MaxString;
    nIGroup: UDINT;
    nIOffset: UDINT;
    nBitSize: UDINT;
    eDataType: E_PLCDataType;
END_STRUCT
END_TYPE
```

Parameter

Name	Typ	Beschreibung
sSymbolName	T_MaxString	Symbolname
sDBSymbolName	T_MaxString	Bezeichnung, die in die Datenbank geschrieben werden soll.
nIGroup	UDINT	Index Group (nur bei ADSRdWrtMode “eADSMode_IGroup_IOffset”)
nIOffset	UDINT	Index Offset (nur bei ADSRdWrtMode “eADSMode_IGroup_IOffset”)
nBitSize	UDINT	Länge in Bits (nur bei ADSRdWrtMode “eADSMode_IGroup_IOffset”)
eDataType	E_PLCDataType [▶ 245]	Datentyp (nur bei ADSRdWrtMode “eADSMode_IGroup_IOffset”)

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.3 Globale Konstanten

6.1.3.1 Konstanten

VAR_GLOBAL CONSTANT GVL

```
AMSPORT_DBSRV : UINT := 21372;
MAX_DBCONNECTIONS : UDINT := 255;
MAX_DBCOLUMNS : UDINT := 255;
MAX_SPPARAMETER : UDINT := 255;
MAX_CONFIGURATIONS : UDINT := 255;
MAX_RECORDS : UDINT := 255;
```

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4 Obsolete

6.1.4.1 Configure Mode

6.1.4.1.1 FB_ConfigTcDBSrv

Funktionsbaustein zum Erstellen, Auslesen und Löschen von Konfigurationseinträgen für den TwinCAT Database Server.

Syntax

Definition:

```
FUNCTION_BLOCK FB_ConfigTcDBSrv
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Create [▶ 161]	Lokal	Erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server
Read [▶ 162]	Lokal	Liest die aktuelle Konfiguration des TwinCAT Database Servers
Delete [▶ 163]	Lokal	Löscht die Datenbank und die AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.1.1.1 Create

Diese Methode erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server. Optional kann ein neuer Eintrag auch nur temporär vom TwinCAT Database Server verwendet werden. Er wird dann nicht in die XML-Datei geschrieben.

Syntax

```
METHOD Create : BOOL
VAR_INPUT
    pTcDBSrvConfig: POINTER TO BYTE;
    cbTcDBSrvConfig: UDINT;
    bTemporary: BOOL := TRUE;
    pConfigID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pTcDBSrvConfig	POINTER TO BYTE	Zeiger der Konfigurationsstruktur, die erzeugt werden soll.
cbTcDBSrvConfig	UDINT	Länge der Konfigurationsstruktur
bTemporary	BOOL	Gibt an, ob die Konfiguration in die XML-Datei gespeichert werden soll.
pConfigID	POINTER TO UDINT	Rückgabezeiger der Konfigurations-ID (hDBID oder hAutoLogGrpID)

Das Anlegen von Autolog-Gruppen wird derzeit nicht unterstützt.

Rückgabewert

Name	Typ	Beschreibung
Create	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    // Any other ConfigType can be used here
    stConfigDB      : T_DBConfig_MsCompactSQL;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Recipes.sdf';

IF fbConfigTcDBSrv.Create(
    pTcDBSrvConfig:= ADR(stConfigDB),
    cbTcDBSrvConfig:= SIZEOF(stConfigDB),
    bTemporary:= TRUE,
    pConfigID:= ADR(myConfigHandle))
THEN
    IF fbSQLStoredProcedure.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.1.1.2 Read

Mit dieser Methode können die aktuellen Konfigurationen des TwinCAT Database Servers gelesen werden. Wenn temporäre Konfigurationen dabei sind, werden diese entsprechend markiert.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    pDBConfig: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    cbDBConfig: UDINT;
    pAutoLogGrpConfig: POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF
ST_ConfigAutoLogGrp;
    cbAutoLogGrpConfig: UDINT;
    pDBCount: POINTER TO UDINT;
    pAutoLogGrpCount: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB [▶ 232]	Pointer-Adresse des Arrays, in das die Datenbankkonfigurationen geschrieben werden sollen.
cbDBConfig	UDINT	Länge des Datenbankkonfiguration-Arrays
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp [▶ 231]	Pointer-Adresse des Arrays, in das die AutoLogGrp Konfigurationen geschrieben werden sollen.
cbAutoLogGrpConfig	UDINT	Länge des AutoLogGrp-Konfiguration-Arrays
pDBCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der Datenbankkonfigurationen gespeichert werden.
pAutoLogGrpCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der AutoLogGrp-Konfigurationen gespeichert werden.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv   : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    aDBConfig         : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    aAutoGrpConfig    : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp;
    nDbCount          : UDINT;
    nAutoGrpCount     : UDINT;
END_VAR
```

```
IF fbConfigTcDBSrv.Read(
    pDBConfig := ADR(aDBConfig),
    cbDBConfig := SIZEOF(aDBConfig),
    pAutologGrpConfig := ADR(aAutoGrpConfig),
    cbAutoLogGrpConfig := SIZEOF(aAutoGrpConfig),
    pDBCount := ADR(nDbCount),
    pAutoLogGrpCount := ADR(nAutoGrpCount))
 THEN
    IF fbConfigTcDBSrv.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.1.1.3 Delete

Mit dieser Methode können Datenbank und AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers gelöscht werden.

Syntax

```
METHOD Delete : BOOL
VAR_INPUT
    eTcDBSrvConfigType: E_TcDBSrvConfigType;
    hConfigID: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
eTcDBSrvConfigType	E_TcDBSrvConfigType	Typ der zu löschenden Konfiguration (Datenbank / AutoLog-Gruppe)
hConfigID	UDINT	ID der zu löschenden Konfiguration (hDBID oder hAutoLogGrpID)

Rückgabewert

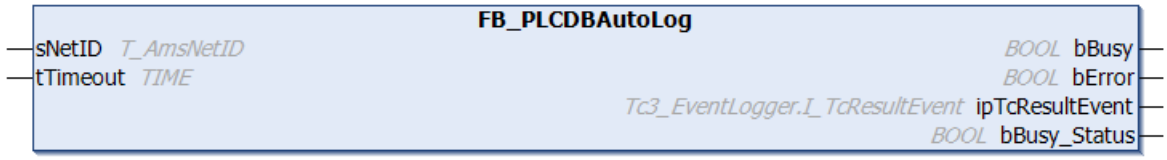
Name	Typ	Beschreibung
Delete	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
END_VAR
```

```
IF fbConfigTcDBSrv.Delete(
    eTcDBSrvConfigType := E_TcDBSrvConfigType.Database,
    hConfigID := myConfigHandle) THEN
IF fbConfigTcDBSrv.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.1.2 FB_PLCDBAutoLog



Funktionsbaustein mit vier Methoden zum Starten und Stoppen von definierten AutoLog-Gruppen, sowie zum Auslesen des entsprechenden Status der Gruppe.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBAutoLog
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
    bBusy_Status: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist, mit Ausnahme der Status Methode.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.
bBusy_Status	BOOL	Die Methode Status kann unabhängig von den anderen drei Methoden des Bausteins ausgeführt werden und hat folglich ein eigenes Busy Flag. Ist TRUE, sobald die Methode Status aktiv ist.

Methoden

Name	Definitionsort	Beschreibung
RunOnce [▶ 165]	Lokal	Führt die AutoLog-Gruppe einmalig aus
Start [▶ 166]	Lokal	Startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen
Status [▶ 166]	Lokal	Fragt den Status der AutoLog-Gruppen ab.
Stop [▶ 167]	Lokal	Stoppt den AutoLog-Modus

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.1.2.1 RunOnce

Mit dieser Methode kann eine AutoLog-Gruppe einmalig ausgeführt werden. Zum Beispiel aufgrund eines Ereignisses in der Steuerung.

Syntax

```
METHOD RunOnce : BOOL
VAR_INPUT
    hAutoLogGrpID: UDINT;
    bAll: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hAutoLogGrpID	UDINT	ID der AutoLog-Gruppe, die einmalig ausgeführt werden soll.
bAll	BOOL	Wenn TRUE, werden alle AutoLog-Gruppen werden einmalig ausgeführt.

Rückgabewert

Name	Typ	Beschreibung
RunOnce	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.RunOnce(hAutologGrpID := 1, bAll := FALSE) THEN
    ; // ...
END_IF
```

6.1.4.1.2.2 Start

Diese Methode startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen.

Syntax

```
METHOD Start : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Start	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Start() THEN
    ; // ...
END_IF
```

6.1.4.1.2.3 Status

Mit dieser Methode kann der Status der AutoLog Gruppen abgefragt werden. Im Rumpf des Bausteins ist ein eigenes Busy Flag für diese Methode vorgesehen, da sie unabhängig von den anderen Methoden des Bausteins aufgerufen werden kann: bBusy_Status.

Syntax

```
METHOD Status : BOOL
VAR_INPUT
    tCheckCycle: TIME;
    pError: POINTER TO BOOL;
    pAutoLogGrpStatus: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
    cbAutoLogGrpStatus: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
tCheckCycle	TIME	Intervallzeit, in der das Statusarray aktualisiert wird.
pError	POINTER TO BOOL	TRUE, wenn ein Fehler beim AutoLog Modus aufgetreten ist.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus	Adresse zum Statusarray, welches alle Gruppen beinhaltet.
cbAutoLogStatus	UDINT	Länge des Statusarrays

Rückgabewert

Name	Typ	Beschreibung
Status	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog(sNetID:='', tTimeout := T#5S);
    bError            : BOOL;
    aAutologGrpStatus : ARRAY[0..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
END_VAR
```

```
IF fbPLCDBAutoLog.Status(tCheckCycle := T#30S, ADR(bError), ADR(aAutologGrpStatus), SIZEOF(aAutologGrpStatus)) THEN
    ; // ...
END_IF
```

6.1.4.1.2.4 Stop

Diese Methode stoppt den AutoLog-Modus.

Syntax

```
METHOD Stop : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Stop	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Stop() THEN
    ; // ...
END_IF
```

6.1.4.2 PLC Expert Mode

6.1.4.2.1 FB_ConfigTcDBSrv

Funktionsbaustein zum Erstellen, Auslesen und Löschen von Konfigurationseinträgen für den TwinCAT Database Server.

Syntax

Definition:

```
FUNCTION_BLOCK FB_ConfigTcDBSrv
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Create [▶ 257]	Lokal	Erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server
Read [▶ 258]	Lokal	Liest die aktuelle Konfiguration des TwinCAT Database Servers
Delete [▶ 259]	Lokal	Löscht die Datenbank und die AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.2.1.1 Create

Diese Methode erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server. Optional kann ein neuer Eintrag auch nur temporär vom TwinCAT Database Server verwendet werden. Er wird dann nicht in die XML-Datei geschrieben.

Syntax

```
METHOD Create : BOOL
VAR_INPUT
    pTcDBSrvConfig: POINTER TO BYTE;
    cbTcDBSrvConfig: UDINT;
```

```
    bTemporary: BOOL := TRUE;
    pConfigID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pTcDBSrvConfig	POINTER TO BYTE	Zeiger der Konfigurationsstruktur, die erzeugt werden soll.
cbTcDBSrvConfig	UDINT	Länge der Konfigurationsstruktur
bTemporary	BOOL	Gibt an, ob die Konfiguration in die XML-Datei gespeichert werden soll.
pConfigID	POINTER TO UDINT	Rückgabezeiger der Konfigurations-ID (hDBID oder hAutoLogGrpID)

Das Anlegen von Autolog-Gruppen wird derzeit nicht unterstützt.

Rückgabewert

Name	Typ	Beschreibung
Create	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    // Any other ConfigType can be used here
    stConfigDB      : T_DBConfig_MsCompactSQL;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Recipes.sdf';

IF fbConfigTcDBSrv.Create(
    pTcDBSrvConfig:= ADR(stConfigDB),
    cbTcDBSrvConfig:= SIZEOF(stConfigDB),
    bTemporary:= TRUE,
    pConfigID:= ADR(myConfigHandle))
THEN
    IF fbSQLStoredProcedure.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.2.1.2 Read

Mit dieser Methode können die aktuellen Konfigurationen des TwinCAT Database Servers gelesen werden. Wenn temporäre Konfigurationen dabei sind, werden diese entsprechend markiert.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    pDBConfig: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    cbDBConfig: UDINT;
    pAutoLogGrpConfig: POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF
ST_ConfigAutoLogGrp;
    cbAutoLogGrpConfig: UDINT;
    pDBCount: POINTER TO UDINT;
    pAutoLogGrpCount: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB [▶ 232]	Pointer-Adresse des Arrays, in das die Datenbankkonfigurationen geschrieben werden sollen.
cbDBConfig	UDINT	Länge des Datenbankkonfiguration-Arrays
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp [▶ 231]	Pointer-Adresse des Arrays, in das die AutoLogGrp Konfigurationen geschrieben werden sollen.
cbAutoLogGrpConfig	UDINT	Länge des AutoLogGrp-Konfiguration-Arrays
pDBCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der Datenbankkonfigurationen gespeichert werden.
pAutoLogGrpCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der AutoLogGrp-Konfigurationen gespeichert werden.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv   : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    aDBConfig         : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    aAutoGrpConfig    : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp;
    nDbCount          : UDINT;
    nAutoGrpCount     : UDINT;
END_VAR
```

```
IF fbConfigTcDBSrv.Read(
    pDBConfig := ADR(aDBConfig),
    cbDBConfig := SIZEOF(aDBConfig),
    pAutologGrpConfig := ADR(aAutoGrpConfig),
    cbAutoLogGrpConfig := SIZEOF(aAutoGrpConfig),
    pDBCount := ADR(nDbCount),
    pAutoLogGrpCount := ADR(nAutoGrpCount))
 THEN
    IF fbConfigTcDBSrv.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.2.1.3 Delete

Mit dieser Methode können Datenbank und AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers gelöscht werden.

Syntax

```
METHOD Delete : BOOL
VAR_INPUT
    eTcDBSrvConfigType: E_TcDBSrvConfigType;
    hConfigID: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
eTcDBSrvConfigType	E_TcDBSrvConfigType	Typ der zu löschenden Konfiguration (Datenbank / AutoLog-Gruppe)
hConfigID	UDINT	ID der zu löschenden Konfiguration (hDBID oder hAutoLogGrpID)

Rückgabewert

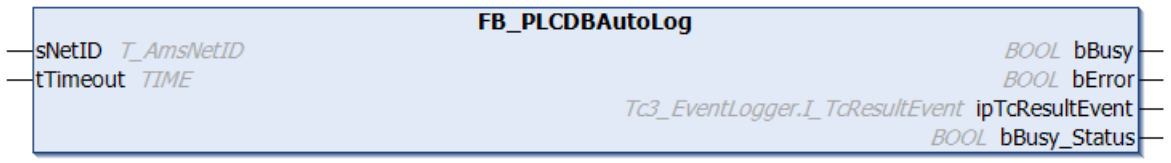
Name	Typ	Beschreibung
Delete	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
END_VAR
```

```
IF fbConfigTcDBSrv.Delete(
    eTcDBSrvConfigType := E_TcDBSrvConfigType.Database,
    hConfigID := myConfigHandle) THEN
IF fbConfigTcDBSrv.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.2.2 FB_PLCDBAutoLog



Funktionsbaustein mit vier Methoden zum Starten und Stoppen von definierten AutoLog-Gruppen, sowie zum Auslesen des entsprechenden Status der Gruppe.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBAutoLog
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
    bBusy_Status: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist, mit Ausnahme der Status Methode.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.
bBusy_Status	BOOL	Die Methode Status kann unabhängig von den anderen drei Methoden des Bausteins ausgeführt werden und hat folglich ein eigenes Busy Flag. Ist TRUE, sobald die Methode Status aktiv ist.

Methoden

Name	Definitionsort	Beschreibung
RunOnce [▶ 165]	Lokal	Führt die AutoLog-Gruppe einmalig aus
Start [▶ 166]	Lokal	Startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen
Status [▶ 166]	Lokal	Fragt den Status der AutoLog-Gruppen ab.
Stop [▶ 167]	Lokal	Stoppt den AutoLog-Modus

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.2.2.1 RunOnce

Mit dieser Methode kann eine AutoLog-Gruppe einmalig ausgeführt werden. Zum Beispiel aufgrund eines Ereignisses in der Steuerung.

Syntax

```
METHOD RunOnce : BOOL
VAR_INPUT
    hAutoLogGrpID: UDINT;
    bAll: BOOL;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hAutoLogGrpID	UDINT	ID der AutoLog-Gruppe, die einmalig ausgeführt werden soll.
bAll	BOOL	Wenn TRUE, werden alle AutoLog-Gruppen werden einmalig ausgeführt.

Rückgabewert

Name	Typ	Beschreibung
RunOnce	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.RunOnce(hAutologGrpID := 1, bAll := FALSE) THEN
    ; // ...
END_IF
```

6.1.4.2.2.2 Start

Diese Methode startet den AutoLog-Modus mit den entsprechenden konfigurierten AutoLog-Gruppen.

Syntax

```
METHOD Start : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Start	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Start() THEN
    ; // ...
END_IF
```

6.1.4.2.2.3 Status

Mit dieser Methode kann der Status der AutoLog Gruppen abgefragt werden. Im Rumpf des Bausteins ist ein eigenes Busy Flag für diese Methode vorgesehen, da sie unabhängig von den anderen Methoden des Bausteins aufgerufen werden kann: bBusy_Status.

Syntax

```
METHOD Status : BOOL
VAR_INPUT
    tCheckCycle: TIME;
    pError: POINTER TO BOOL;
    pAutoLogGrpStatus: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
    cbAutoLogGrpStatus: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
tCheckCycle	TIME	Intervallzeit, in der das Statusarray aktualisiert wird.
pError	POINTER TO BOOL	TRUE, wenn ein Fehler beim AutoLog Modus aufgetreten ist.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus	Adresse zum Statusarray, welches alle Gruppen beinhaltet.
cbAutoLogStatus	UDINT	Länge des Statusarrays

Rückgabewert

Name	Typ	Beschreibung
Status	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog(sNetID:='', tTimeout := T#5S);
    bError            : BOOL;
    aAutologGrpStatus : ARRAY[0..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus;
END_VAR
```

```
IF fbPLCDBAutoLog.Status(tCheckCycle := T#30S, ADR(bError), ADR(aAutologGrpStatus), SIZEOF(aAutologGrpStatus)) THEN
    ; // ...
END_IF
```

6.1.4.2.2.4 Stop

Diese Methode stoppt den AutoLog-Modus.

Syntax

```
METHOD Stop : BOOL
```

Rückgabewert

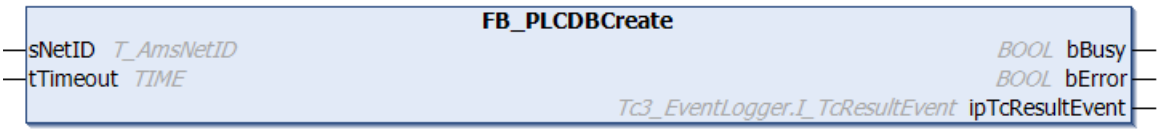
Name	Typ	Beschreibung
Stop	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBAutoLog    : FB_PLCDBAutoLog (sNetID:='', tTimeout := T#5S);
END_VAR
```

```
IF fbPLCDBAutoLog.Stop() THEN
    ; // ...
END_IF
```

6.1.4.2.3 FB_PLCDBCreate



Funktionsbaustein mit zwei Methoden. Mit der einen Methode kann er Datenbanken aus der SPS heraus auf einem in der SPS angegebenen Datenbank-Server erstellen. Mit der anderen Methode kann er in einer angegebenen Datenbank eine neue Tabelle erzeugen.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBCreate
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Database [▶ 264]	Lokal	Erzeugt eine neue Datenbank
Table [▶ 265]	Lokal	Erzeugt eine neue Tabelle mit der Tabellenstruktur, die über ein Array mit x Elementen bzw. x Spalten in der SPS definiert wird.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.2.3.1 Database

Diese Methode erzeugt eine neue Datenbank. Optional kann angegeben werden, ob die erzeugte Datenbank auch für die Konfiguration des TwinCAT Database Servers übernommen werden soll.

Syntax

```
METHOD Database : BOOL
VAR_INPUT
    pDatabaseConfig: POINTER TO BYTE;
    cbDatabaseConfig: UDINT;
    bCreateXMLConfig: BOOL;
    pDBID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDatabaseConfig	POINTER TO BYTE	Adresse zu der Datenbankkonfigurationsstruktur [▶ 232]
cbDatabaseConfig	UDINT	Länge der Datenbankkonfigurationsstruktur
bCreateXMLConfig	BOOL	Gibt an, ob die neu erzeugte Datenbank als neuer Konfigurationseintrag in die XML-Datei geschrieben werden soll.
pDBID	UDINT	Gibt die hDBID zurück, wenn ein neuer Konfigurationseintrag erzeugt wurde.

Rückgabewert

Name	Typ	Beschreibung
Database	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBCreate : FB_PLCDBCreateEvt(sNetID := '', tTimeout := T#5S);
    stConfigDB    : T_DBConfig_MsCompactSQL;
    hDBID         : UDINT;
    tcMessage     : I_TcMessage;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Test.sdf';

IF fbPLCDBCreate.Database(
    pDatabaseConfig:= ADR(stConfigDB),
    cbDatabaseConfig := SIZEOF(stConfigDB),
    bCreateXMLConfig := TRUE,
    pDBID := ADR(hDBID))
THEN
    IF fbPLCDBCreate.bError THEN
        tcMessage := fbPLCDBCreate.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.2.3.2 Table

Diese Methode erzeugt eine neue Tabelle mit der Tabellenstruktur, die über ein Array mit x Elementen bzw. x Spalten in der SPS definiert wird.

Syntax

```
METHOD Table : BOOL
VAR_INPUT
    hDBID : UDINT;
    sTableName : T_MaxString;
    pTableCfg : POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ColumnInfo;
    cbTableCfg : UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	MaxString	Name der Tabelle, die erzeugt werden soll.
pTableCfg	POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ColumnInfo [▶ 247]	Gibt die Pointer-Adresse des Tabellenstrukturarrays an. In diesem Array werden die einzelnen Spalten beschrieben.
cbTableCfg	UDINT	Gibt die Länge der Arrays an, in dem die Spalten konfiguriert sind.

Rückgabewert

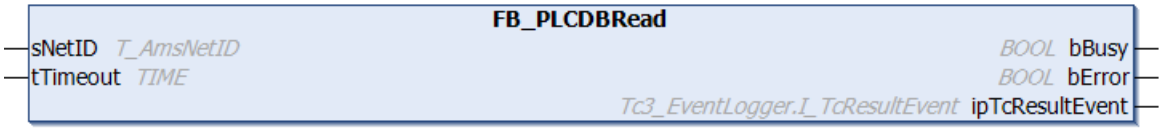
Name	Typ	Beschreibung
Table	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBCreate : FB_PLCDBCreateEvt(sNetID := '', tTimeout := T#5S);
    ColumnInfo    : ARRAY [0..14] OF ST_ColumnInfo;
    tcMessage     : I_TcMessage;
END_VAR
```

```
ColumnInfo[0].sName := 'colBigInt';     ColumnInfo[0].eType := E_ColumnType.BigInt;     ColumnInfo[0
].nLength := 8;     ColumnInfo[0].sProperty := 'IDENTITY(1,1)';
ColumnInfo[1].sName := 'colInteger';    ColumnInfo[1].eType := E_ColumnType.Integer;    ColumnInfo[1
].nLength := 4;
ColumnInfo[2].sName := 'colSmallInt';   ColumnInfo[2].eType := E_ColumnType.SmallInt;   ColumnInfo[2
].nLength := 2;
ColumnInfo[3].sName := 'colTinyInt';    ColumnInfo[3].eType := E_ColumnType.TinyInt;    ColumnInfo[3
].nLength := 1;
ColumnInfo[4].sName := 'colBit';        ColumnInfo[4].eType := E_ColumnType.BIT_;       ColumnInfo[4
].nLength := 1;
ColumnInfo[5].sName := 'colMoney';      ColumnInfo[5].eType := E_ColumnType.Money;      ColumnInfo[5
].nLength := 8;
ColumnInfo[6].sName := 'colFloat';      ColumnInfo[6].eType := E_ColumnType.Float;      ColumnInfo[6
].nLength := 8;
ColumnInfo[7].sName := 'colReal';       ColumnInfo[7].eType := E_ColumnType.REAL_;      ColumnInfo[7
].nLength := 4;
ColumnInfo[8].sName := 'colDateTime';   ColumnInfo[8].eType := E_ColumnType.DateTime;   ColumnInfo[8
].nLength := 4;
ColumnInfo[9].sName := 'colNText';      ColumnInfo[9].eType := E_ColumnType.NText;      ColumnInfo[9
].nLength := 256;
ColumnInfo[10].sName := 'colNChar';     ColumnInfo[10].eType := E_ColumnType.NChar;     ColumnInfo[1
0].nLength := 10;
ColumnInfo[11].sName := 'colImage';     ColumnInfo[11].eType := E_ColumnType.Image;     ColumnInfo[1
1].nLength := 256;
ColumnInfo[12].sName := 'colNVarChar';  ColumnInfo[12].eType := E_ColumnType.NVarChar;  ColumnInfo[1
2].nLength := 50;
ColumnInfo[13].sName := 'colBinary';    ColumnInfo[13].eType := E_ColumnType.Binary;    ColumnInfo[1
3].nLength := 30;
ColumnInfo[14].sName := 'colVarBinary'; ColumnInfo[14].eType := E_ColumnType.VarBinary; ColumnInfo[1
4].nLength := 20;

IF fbPLCDBCreate.Table(
    hDBID:= 1,
    sTableName:= 'myNewTable',
    pTableCfg:= ADR(ColumnInfo),
    cbTableCfg:= SIZEOF(ColumnInfo))
THEN
    IF fbPLCDBCreate.bError THEN
        TcMessage:= fbPLCDBCreate.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.2.4 FB_PLCDBRead



Funktionsbaustein zum Auslesen von Datensätzen aus einer Datenbank.

Syntax

```
FUNCTION_BLOCK FB_PLCDBRead
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Type	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Type	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Read [▶ 267]	Lokal	Liest eine vorgegebene Anzahl an Datensätze aus einer Datenbanktabelle mit der von Beckhoff vorgegebenen Standardtabellenstruktur.
ReadStruct [▶ 269]	Lokal	Liest eine vorgegebene Anzahl an Datensätze aus einer Datenbanktabelle mit beliebiger Tabellenstruktur.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.2.4.1 Read

Diese Methode liest eine vorgegebene Anzahl an Datensätze aus einer Datenbanktabelle mit der von Beckhoff vorgegebenen Standardtabellenstruktur. Die Standardtabellenstruktur wird unter anderem beim AutoLog-Modus und beim FB_DBWrite-Baustein verwendet.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    sDBSymbolName: T_MaxString;
    eOrderBy: E_OrderColumn := E_OrderColumn.eColumnID;
    eOrderType: E_OrderType := E_OrderType.eOrder_ASC;
    nStartIndex: UDINT;
    nRecordCount: UDINT;
    pData: POINTER TO ST_StandardRecord;
    cbData: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
sDBSymbolName	T_MaxString	Symbolname, der in der Standardtabellenstruktur ausgelesen werden soll.
eOrderBy	E_OrderColumn.eColumnID	Sortierspalte (ID, Timestamp, Name oder Value)
eOrderType	E_OrderType.eOrder_ASC	Sortierrichtung (ASC oder DESC)
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pData	POINTER TO ST_StandardRecord	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Strukturarrays in Byte an.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBRead    : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
    ReadStruct     : ST_StandardRecord;
    tcMessage      : I_TcMessage;
END_VAR
```

```
IF fbPLCDBRead.Read(
    hDBID:= 1,
    sTableName:= 'MyTable_WithLReal',
    sDBSymbolName:= 'MyValue',
    eOrderBy:= E_OrderColumn.ID,
    eOrderType:= E_OrderType.DESC,
    nStartIndex:= 0,
    nRecordCount:= 1,
    pData:= ADR(ReadStruct),
    cbData:= SIZEOF(ReadStruct))
THEN
    IF fbPLCDBRead.bError THEN
        tcMessage := fbPLCDBRead.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der SPS:

Expression	Type	Value
ReadStruct	ST_StandardRecord	
nID	LINT	2
dtTimestamp	DATE_AND_TIME	DT#2018-2-1-16:8:8
sName	STRING(80)	'MyValue'
rValue	LREAL	15.9

6.1.4.2.4.2 ReadStruct

Diese Methode liest eine vorgegebene Anzahl an Datensätze aus einer Datenbanktabelle mit beliebiger Tabellenstruktur.

Syntax

```
METHOD ReadStruct : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
    cbColumnNames: UDINT;
    sOrderByColumn: STRING(50);
    eOrderType: E_OrderType := E_OrderType.eOrder_ASC
    nStartIndex: UDINT;
    nRecordCount: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Adresse des Arrays, welches die zu lesenden Spaltennamen beinhaltet.
cbColumnNames	UDINT	Länge des Spaltennamen-Arrays
sOrderByColumn	STRING(50)	Name der Sortierspalte
eOrderType	E_OrderType	Sortierrichtung (ASC oder DESC)
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pData	POINTER TO BYTE	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Strukturarrays in Byte an.

Rückgabewert

Name	Typ	Beschreibung
ReadStruct	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbPLCDBRead    : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
    myCustomStruct : ST_Record;
    tcMessage      : I_TcMessage;
END_VAR
```

```
TYPE ST_Record :
STRUCT
    nID        : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName      : STRING;
    nSensor1   : LREAL;
    nSensor2   : LREAL;
END_STRUCT
END_TYPE
```

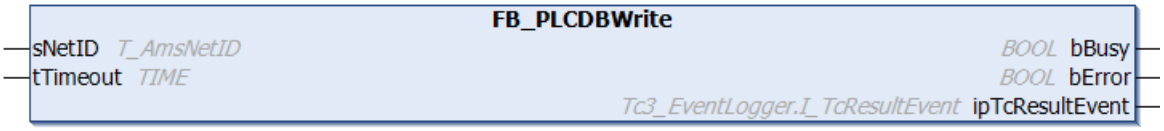
```
// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';
ColumnNames[3] := 'Sensor1';
```

```
ColumnNames[4] := 'Sensor2';

IF fbPLCDBRead.ReadStruct(
    hDBID:= 1,
    sTableName:= 'MyTable_Struct',
    pColumnNames:= ADR(ColumnNames),
    cbColumnNames:= SIZEOF(ColumnNames),
    sOrderByColumn:= ColumnNames[0],
    eOrderType:= E_OrderType.DESC,
    nStartIndex:= 0,
    nRecordCount:= 1,
    pData:= ADR(myCustomStruct),
    cbData:= SIZEOF(myCustomStruct))
THEN
    IF fbPLCDBRead.bError THEN
        tcMessage:= fbPLCDBRead.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der SPS:

Expression	Type	Value
myCustomStruct	ST_Record	
nID	LINT	1
dtTimestamp	DATE_AND_TIME	DT#2018-2-1-15:17:54
sName	STRING	'MyStructVal'
nSensor1	LREAL	12.34
nSensor2	LREAL	102.5

6.1.4.2.5 FB_PLCDBWrite



Funktionsbaustein zum Schreiben von Datensätzen in eine Datenbank.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBWrite
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Write [▶ 271]	Lokal	Erzeugt einen Datensatz in der von Beckhoff vorgegebenen Standardtabellenstruktur.
WriteBySymbol [▶ 272]	Lokal	Liest den Wert eines vorgegebenen ADS-Symbols aus und speichert diesen in der von Beckhoff vorgegebenen Standardtabellenstruktur ab.
WriteStruct [▶ 274]	Lokal	Erzeugt einen Datensatz mit einer beliebigen Tabellenstruktur

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.2.5.1 Write

Diese Methode erzeugt einen Datensatz in der von Beckhoff vorgegebenen Standardtabellenstruktur.

Syntax

```
METHOD Write : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pValue: POINTER TO BYTE;
    cbValue: UDINT;
    sDBSymbolName: T_MaxString;
    eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
    nRingBuffParameter: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
pValue	POINTER TO BYTE	Adresse der Variable, die in die Standardtabellenstruktur geloggt werden soll.
cbValue	UDINT	Länge der Variable, die geloggt werden soll.
sDBSymbolName	T_MaxString	Name, der in die Tabelle mit geloggt wird.
eDBWriteMode	E_WriteMode	Gibt den Schreibmodus an. (Anhängen, updaten, Ringbuffer)
nRingBuffParameter	UDINT	Zusatzparameter für den Schreibmodus "Ringbuffer".

Rückgabewert

Name	Typ	Beschreibung
Write	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Dieses Beispiel zeigt die Verwendung der Methode FB_PLCDBWriteEvt.Write:

```
VAR
    fbPLCDBWrite    : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
    myValue         : LREAL := 43.23;
    tcMessage       : I_TcMessage;
END_VAR
```

```
IF fbPLCDBWrite.Write(
    hDBID:= 1,
    sTableName:= 'myTable_WithLReal',
    pValue:= ADR(myValue),
    cbValue:= SIZEOF(myValue),
    sDBSymbolName:= 'MyValue',
    eDBWriteMode:= E_WriteMode.eADS_TO_DB_RingBuff_Count,
    nRingBuffParameter:= 3)
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der Datenbank:

ID	Timestamp	Name	Value
27	Has been dropped		
28	'2018-01-30 14:04:19'	'MyValue'	41.23
29	'2018-01-30 14:04:29'	'MyValue'	42.23
30	'2018-01-30 14:04:39'	'MyValue'	43.23

Durch die Ringbufferoption werden immer nur 3 Einträge dieses Namens in der Datenbank sein. Ältere werden gelöscht.

6.1.4.2.5.2 WriteBySymbol

Diese Methode liest den Wert eines vorgegebenen ADS-Symbols aus und speichert diesen in der von Beckhoff vorgegebenen Standardtabellenstruktur ab. Es können auch ADS-Symbole von anderen ADS-Geräten ausgelesen werden.

Syntax

```
METHOD WriteBySymbol : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    stADSDevice: ST_ADSDevice;
    stSymbol: ST_Symbol;
    eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
    nRingBuffParameter: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
stADSDevice	ST_ADSDevice	ADS-Gerät, von dem ein Symbol in die Standardtabellenstruktur geloggt werden soll.
stSymbol	ST_Symbol	Symbolname der zu schreibenden Variablen
eDBWriteMode	E_WriteMode	Gibt den Schreibmodus an. (Anhängen, updaten, Ringbuffer)
nRingBuffParameter	UDINT	Zusatzparameter für den Schreibmodus “Ringbuffer“

Rückgabewert

Name	Typ	Beschreibung
WriteBySymbol	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Dieses Beispiel zeigt die Verwendung der Methode FB_PLCDBWriteEvt.WriteBySymbol:

```
VAR
    fbPLCDBWrite    :  FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
    myValue         :  LREAL := 43.23;
    myAdsDevice     :  ST_ADSDevice;
    mySymbol        :  ST_Symbol;
    tcMessage       :  I_TcMessage;
END_VAR
```

```
// Set ADSDevice Information
myAdsDevice.sDevNetID     := '127.0.0.1.1.1';
myAdsDevice.nDevPort      := 851;
myAdsDevice.eADSRdWrtMode := E_ADSRdWrtMode.bySymbolName;
myAdsDevice.tTimeout      := T#5S;

// Set Symbol Information
mySymbol.eDataType        := E_PLCDataType.eType_LREAL;
mySymbol.sDBSymbolName    := 'MySymbol';
mySymbol.sSymbolName      := 'MAIN.myValue';
mySymbol.nBitSize         := 8;

// Call Functionblock
IF fbPLCDBWrite.WriteBySymbol(
    hDBID:= 1,
    sTableName:= 'myTable_WithLReal',
    stADSDevice:= myAdsDevice,
    stSymbol:= mySymbol,
    eDBWriteMode:= E_WriteMode.eADS_TO_DB_Append,
    nRingBuffParameter:= 1)
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der Datenbank:

ID	Timestamp	Name	Value
28	'2018-01-30 14:04:19'	'MyValue'	41.23
29	'2018-01-30 14:04:29'	'MyValue'	42.23
30	'2018-01-30 14:04:39'	'MyValue'	43.23
31	'2018-01-30 14:06:12’	‘MySymbol’	86.2

6.1.4.2.5.3 WriteStruct

Diese Methode erzeugt einen Datensatz mit einer beliebigen Tabellenstruktur.

Syntax

```
METHOD WriteStruct : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pRecord: POINTER TO BYTE;
    cbRecord: UDINT;
    pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
    cbColumnNames: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
sTableName	T_MaxString	Name der Tabelle, aus der gelesen werden soll.
pRecord	POINTER TO BYTE	Adresse einer Struktur, die in eine beliebige Tabellenstruktur geloggt werden soll.
cbRecord	UDINT	Länge der zu schreibenden Struktur
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Adresse des Arrays, welches die zu füllenden Spaltennamen beinhaltet.
cbColumnNames	UDINT	Länge des Spaltennamen-Arrays

Rückgabewert

Name	Typ	Beschreibung
WriteStruct	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Dieses Beispiel zeigt die Verwendung der Methode FB_PLCDBWriteEvt.WriteStruct:

```
VAR
    fbPLCDBWrite    :  FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
    myRecord        :  ST_Record;
    ColumnNames     :  ARRAY[0..4] OF STRING(50);

    systime         :  GETSYSTEMTIME;
    currentTime     :  T_FILETIME;
    tcMessage       :  I_TcMessage;
END_VAR
```

```
TYPE ST_Record :
STRUCT
    nID        : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName      : STRING;
    nSensor1   : LREAL;
    nSensor2   : LREAL;
END_STRUCT
END_TYPE
```

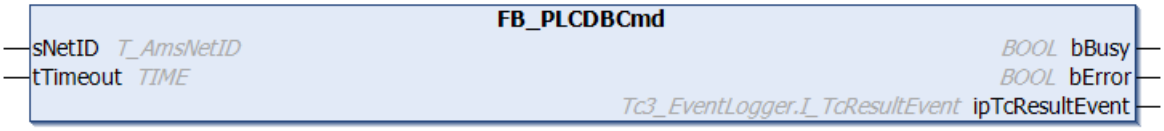
```
// set Values
systime(timeLoDw => currentTime.dwLowDateTime, timeHiDW => currentTime.dwHighDateTime );
myRecord.dtTimestamp := FILETIME_TO_DT(currentTime);
myRecord.sName       := 'MyStructVal';
myRecord.nSensor1    := 12.34;
myRecord.nSensor2    := 102.5;

// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
```

```
ColumnNames[2] := 'Name';
ColumnNames[3] := 'Sensor1';
ColumnNames[4] := 'Sensor2';

// Call Functionblock
IF fbPLCDBWrite.WriteStruct(
    hDBID:= 1,
    sTableName:= 'myTable_Struct',
    pRecord:= ADR(myRecord),
    cbRecord:= SIZEOF(myRecord),
    pColumnNames:= ADR(ColumnNames) ,
    cbColumnNames:= SIZEOF(ColumnNames))
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Ergebnis in der Datenbank:

ID	Timestamp	Name	Sensor1	Sensor2
5	'2018-01-30 15:23:26'	‘MyStructVal’	12.34	102.5

6.1.4.2.6 FB_PLCDBCmd



Funktionsbaustein mit zwei Methoden. Eigene SQL-Kommandos können definiert und übergeben werden. Platzhalter im SQL-Kommando können mit Strukturen in der SPS korrelieren, welche die Tabellenstruktur wiederspiegeln. Der Database Server setzt letztendlich die aktuellen Daten der Struktur in das SQL-Kommando ein.

Syntax

Definition:

```
FUNCTION_BLOCK FB_PLCDBCmd
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Execute [▶ 276]	Lokal	Sendet beliebige SQL-Kommandos an die Datenbank gesendet werden. Zurückgelieferte Datensätze können nicht ausgelesen werden.
ExecuteDataReturn [▶ 277]	Lokal	Sendet beliebige SQL-Kommandos an die Datenbank gesendet werden. Eine vorgegebene Anzahl von Datensätze kann ausgelesen werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.2.6.1 Execute

Mit dieser Methode können beliebige SQL-Kommandos an die Datenbank gesendet werden. Die Datenbankverbindung wird bei jedem Aufruf geöffnet und nach der Ausführung wieder geschlossen. Es besteht die Möglichkeit, Platzhalter im Kommando zu definieren, die dann vom TwinCAT Database Server vor der Ausführung mit den entsprechenden Werten befüllt werden. Zurückgelieferte Datensätze können nicht ausgelesen werden.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    hDBID: UDINT;
    pExpression: POINTER TO BYTE;
    cbExpression: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
    cbParameter: UDINT;
END_VAR
```

Beispiel

```
VAR
    fbPLCDBCmd    : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
    sCmd          : STRING (1000);
    myStruct      : ST_DataAll;
    aPara         : ARRAY[0..14] OF ST_ExpParameter;
    tcMessage     : I_TcMessage;
END_VAR
```

```
TYPE ST_DataAll :
STRUCT
    colBigInt: LINT;
    colInteger: DINT;
    colSmallInt: INT;
    colTinyInt: BYTE;
    colBit: BOOL;
    colMoney: LREAL;
    colFloat: LREAL;
    colReal: REAL;
```

```
    colDateTime: DT;
    colNText: STRING(255);
    colNChar: STRING(10);
    colImage: ARRAY[0..255] OF BYTE;
    colNVarChar: STRING(50);
    colBinary: ARRAY[0..29] OF BYTE;
    colVarBinary: ARRAY[0..19] OF BYTE;
END_STRUCT
END_TYPE
```

```
// set Parameter configuration
aPara[0].sParaName := 'colBigInt';    aPara[0].eParaType :=
E_ExpParameterType.Int64;      aPara[0].nParaSize := 8;
aPara[1].sParaName := 'colInteger';   aPara[1].eParaType :=
E_ExpParameterType.Int32;      aPara[1].nParaSize := 4;
aPara[2].sParaName := 'colSmallInt';  aPara[2].eParaType :=
E_ExpParameterType.Int16;      aPara[2].nParaSize := 2;
aPara[3].sParaName := 'colTinyInt';   aPara[3].eParaType :=
E_ExpParameterType.Byte_;      aPara[3].nParaSize := 1;
aPara[4].sParaName := 'colBit';       aPara[4].eParaType :=
E_ExpParameterType.Boolean;    aPara[4].nParaSize := 1;
aPara[5].sParaName := 'colMoney';     aPara[5].eParaType :=
E_ExpParameterType.Double64;   aPara[5].nParaSize := 8;
aPara[6].sParaName := 'colFloat';     aPara[6].eParaType :=
E_ExpParameterType.Double64;   aPara[6].nParaSize := 8;
aPara[7].sParaName := 'colReal';      aPara[7].eParaType :=
E_ExpParameterType.Float32;    aPara[7].nParaSize := 4;
aPara[8].sParaName := 'colDateTime';  aPara[8].eParaType :=
E_ExpParameterType.DateTime;   aPara[8].nParaSize := 4;
aPara[9].sParaName := 'colNText';     aPara[9].eParaType :=
E_ExpParameterType.STRING_;    aPara[9].nParaSize := 256;
aPara[10].sParaName:= 'colNChar';     aPara[10].eParaType :=
E_ExpParameterType.STRING_;   aPara[10].nParaSize := 10;
aPara[11].sParaName:= 'colImage';     aPara[11].eParaType :=
E_ExpParameterType.ByteArray; aPara[11].nParaSize := 256;
aPara[12].sParaName:= 'colNVarChar';  aPara[12].eParaType :=
E_ExpParameterType.STRING_;   aPara[12].nParaSize := 50;
aPara[13].sParaName:= 'colBinary';    aPara[13].eParaType :=
E_ExpParameterType.ByteArray; aPara[13].nParaSize := 30;
aPara[14].sParaName:= 'colVarBinary'; aPara[14].eParaType :=
E_ExpParameterType.ByteArray; aPara[14].nParaSize := 20;

// set command
sCmd := 'INSERT INTO MyTableName (colInteger, colSmallInt, colTinyInt, colBit, colMoney, colFloat,
colReal, colDateTime, colNText, colNChar, colImage, colNVarChar, colBinary, colVarBinary) VALUES
({colInteger}, {colSmallInt}, {colTinyInt}, {colBit}, {colMoney}, {colFloat}, {colReal},
{colDateTime}, {colNText}, {colNChar}, {colImage}, {colNVarChar}, {colBinary}, {colVarBinary})';

// call functionblock
IF fbPLCDBCmd.Execute(
    hDBID:= 1,
    pExpression:= ADR(sCmd),
    cbExpression:= SIZEOF(sCmd),
    pData:= ADR(myStruct),
    cbData:= SIZEOF(myStruct),
    pParameter:= ADR(aPara),
    cbParameter:= SIZEOF(aPara))
THEN
    IF fbPLCDBCmd.bError THEN
        tcMessage := fbPLCDBCmd.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.2.6.2 ExecuteDataReturn

Mit dieser Methode können beliebige SQL-Kommandos an die Datenbank gesendet werden. Die Datenbankverbindung wird bei jedem Aufruf geöffnet und nach der Ausführung wieder geschlossen. Es besteht die Möglichkeit, Platzhalter im Kommando zu definieren, die dann vom TwinCAT Database Server vor der Ausführung mit den entsprechenden Werten befüllt werden. Eine vorgegebene Anzahl von Datensätzen kann ausgelesen werden.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    hDBID: UDINT;
    pExpression: POINTER TO BYTE;
    cbExpression: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
    cbParameter: UDINT;
    nStartIndex: UDINT;
    nRecordCount: UDINT;
    pReturnData: POINTER TO BYTE;
    cbReturnData: UDINT;
    pRecords: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.
pExpression	POINTER TO BYTE	Adresse der String-Variablen mit dem SQL Kommando.
cbExpression	UDINT	Länge der String-Variablen mit dem SQL-Kommando.
pData	POINTER TO BYTE	Adresse der Struktur mit den Parameterwerten
cbData	UDINT	Länge der Struktur mit den Parameterwerten
pParameter	POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter	Adresse des Strukturarrays mit den Parameterinformationen.
cbParameter	UDINT	Länge des Strukturarrays mit den Parameterinformationen.
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pReturnData	POINTER TO BYTE	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbReturnData	UDINT	Gibt die Größe des Strukturarrays in Byte an.
pRecords	POINTER TO BYTE	Anzahl der ausgelesenen Datensätze.

Rückgabewert

Name	Typ	Beschreibung
ExecuteDataReturn	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Parametrieren des Kommandos

Die Spaltennamen für die einzelnen Parameter werden im SQL-Kommando in geschweiften Klammern angegeben.
Beispiel: ‚SELECT * FROM MyHouse_Temperatures WHERE Room = {SelectedRoom}'.
SelectedRoom muss dabei in der Struktur ST_ExpParameter entsprechend als Parametername angegeben sein.

Einige Datenbanken unterstützen das Parametrieren von SQL Klauseln nicht. (TOP/LIMIT/ROWNUM/…) Auch parametrierbare Tabellennamen werden in der Regel nicht unterstützt.

Beispiel

```
VAR
    fbPLCDBCmd       : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
    sCmd             : STRING (1000);
    stPara           : ST_ExpParameter;
    RecordAmt        : ULINT := 3;
    ReturnDataStruct : ARRAY [0..9] OF ST_DataAll;
```

```
    nRecords         : UDINT;
    tcMessage        : I_TcMessage;
END_VAR
```

```
// set Parameter configuration
stPara.eParaType := E_ExpParameterType.Int64;
stPara.nParaSize := 8;
stPara.sParaName := 'RecordAmt';

// set command with placeholder
sCmd := 'SELECT TOP ({RecordAmt}) * FROM MyTableName';

// call functionblock
IF fbPLCDBCmd.ExecuteDataReturn(
    hDBID:= 1,
    pExpression:= ADR(sCmd),
    cbExpression:= SIZEOF(sCmd),
    pData:= ADR(RecordAmt),
    cbData:= SIZEOF(RecordAmt),
    pParameter:= ADR(stPara),
    cbParameter:= SIZEOF(stPara),
    nStartIndex:= 0,
    nRecordCount:= 10,
    pReturnData:= ADR(ReturnDataStruct),
    cbReturnData:= SIZEOF(ReturnDataStruct),
    pRecords:= ADR(nRecords))
THEN
    IF fbPLCDBCmd.bError THEN
        tcMessage := fbPLCDBCmd.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.3 SQL Expert Mode

6.1.4.3.1 FB_ConfigTcDBSrv

Funktionsbaustein zum Erstellen, Auslesen und Löschen von Konfigurationseinträgen für den TwinCAT Database Server.

Syntax

Definition:

```
FUNCTION_BLOCK FB_ConfigTcDBSrv
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Create [▶ 161]	Lokal	Erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server
Read [▶ 162]	Lokal	Liest die aktuelle Konfiguration des TwinCAT Database Servers
Delete [▶ 163]	Lokal	Löscht die Datenbank und die AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.3.1.1 Create

Diese Methode erzeugt neue Einträge in der XML-Konfigurationsdatei für den TwinCAT Database Server. Optional kann ein neuer Eintrag auch nur temporär vom TwinCAT Database Server verwendet werden. Er wird dann nicht in die XML-Datei geschrieben.

Syntax

```
METHOD Create : BOOL
VAR_INPUT
    pTcDBSrvConfig: POINTER TO BYTE;
    cbTcDBSrvConfig: UDINT;
    bTemporary: BOOL := TRUE;
    pConfigID: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pTcDBSrvConfig	POINTER TO BYTE	Zeiger der Konfigurationsstruktur, die erzeugt werden soll.
cbTcDBSrvConfig	UDINT	Länge der Konfigurationsstruktur
bTemporary	BOOL	Gibt an, ob die Konfiguration in die XML-Datei gespeichert werden soll.
pConfigID	POINTER TO UDINT	Rückgabezeiger der Konfigurations-ID (hDBID oder hAutoLogGrpID)

Das Anlegen von Autolog-Gruppen wird derzeit nicht unterstützt.

Rückgabewert

Name	Typ	Beschreibung
Create	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
    // Any other ConfigType can be used here
    stConfigDB      : T_DBConfig_MsCompactSQL;
END_VAR
```

```
stConfigDB.bAuthentification := FALSE;
stConfigDB.sServer := 'C:\Recipes.sdf';

IF fbConfigTcDBSrv.Create(
    pTcDBSrvConfig:= ADR(stConfigDB),
    cbTcDBSrvConfig:= SIZEOF(stConfigDB),
    bTemporary:= TRUE,
    pConfigID:= ADR(myConfigHandle))
THEN
    IF fbSQLStoredProcedure.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.3.1.2 Read

Mit dieser Methode können die aktuellen Konfigurationen des TwinCAT Database Servers gelesen werden. Wenn temporäre Konfigurationen dabei sind, werden diese entsprechend markiert.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    pDBConfig: POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    cbDBConfig: UDINT;
    pAutoLogGrpConfig: POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF
ST_ConfigAutoLogGrp;
```

```
    cbAutoLogGrpConfig: UDINT;
    pDBCount: POINTER TO UDINT;
    pAutoLogGrpCount: POINTER TO UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB [▶ 232]	Pointer-Adresse des Arrays, in das die Datenbankkonfigurationen geschrieben werden sollen.
cbDBConfig	UDINT	Länge des Datenbankkonfiguration-Arrays
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp [▶ 231]	Pointer-Adresse des Arrays, in das die AutoLogGrp Konfigurationen geschrieben werden sollen.
cbAutoLogGrpConfig	UDINT	Länge des AutoLogGrp-Konfiguration-Arrays
pDBCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der Datenbankkonfigurationen gespeichert werden.
pAutoLogGrpCount	POINTER TO UDINT	Pointer-Adresse, wo die Anzahl der AutoLogGrp-Konfigurationen gespeichert werden.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv   : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    aDBConfig         : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigDB;
    aAutoGrpConfig    : ARRAY[0..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp;
    nDbCount          : UDINT;
    nAutoGrpCount     : UDINT;
END_VAR
```

```
IF fbConfigTcDBSrv.Read(
    pDBConfig := ADR(aDBConfig),
    cbDBConfig := SIZEOF(aDBConfig),
    pAutologGrpConfig := ADR(aAutoGrpConfig),
    cbAutoLogGrpConfig := SIZEOF(aAutoGrpConfig),
    pDBCount := ADR(nDbCount),
    pAutoLogGrpCount := ADR(nAutoGrpCount))
 THEN
    IF fbConfigTcDBSrv.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.3.1.3 Delete

Mit dieser Methode können Datenbank und AutoLog-Gruppen aus der Konfiguration des TwinCAT Database Servers gelöscht werden.

Syntax

```
METHOD Delete : BOOL
VAR_INPUT
    eTcDBSrvConfigType: E_TcDBSrvConfigType;
    hConfigID: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
eTcDBSrvConfigType	E_TcDBSrvConfigType	Typ der zu löschenden Konfiguration (Datenbank / AutoLog-Gruppe)
hConfigID	UDINT	ID der zu löschenden Konfiguration (hDBID oder hAutoLogGrpID)

Rückgabewert

Name	Typ	Beschreibung
Delete	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    myConfigHandle  : INT;
END_VAR
```

```
IF fbConfigTcDBSrv.Delete(
    eTcDBSrvConfigType := E_TcDBSrvConfigType.Database,
    hConfigID := myConfigHandle) THEN
IF fbConfigTcDBSrv.bError THEN
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

6.1.4.3.2 FB_SQLDatabase

Funktionsbaustein zum Öffnen, Schließen und Verwalten einer Datenbankverbindung.

Syntax

Definition:

```
FUNCTION_BLOCK FB_SQLDatabase
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Connect [▶ 284]	Lokal	Öffnet eine Verbindung zu einer deklarierten Datenbank.
CreateCmd [▶ 285]	Lokal	Initialisiert eine Instanz des Bausteins FB_SQLCommand mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabase.
CreateSP [▶ 285]	Lokal	Initialisiert eine Instanz des Bausteins FB_SQLStoredProcedure mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabase Bausteins.
Disconnect [▶ 286]	Lokal	Schließt die Verbindung zur Datenbank, die von dieser Bausteininstanz geöffnet wurde.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.3.2.1 Connect

Diese Methode öffnet eine Verbindung zu einer deklarierten Datenbank.

Syntax

```
METHOD Connect : BOOL
VAR_INPUT
    hDBID: UDINT := 1;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
hDBID	UDINT	Gibt die ID der zu verwendenden Datenbank an.

Rückgabewert

Name	Typ	Beschreibung
Connect	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR
```

```
// open connection
IF fbSqlDatabase.Connect(1) THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
    ELSE
```

```
        nState := nState+1;
    END_IF
END_IF
```

6.1.4.3.2.2 CreateCmd

Mit dieser Methode wird eine Instanz des Bausteins FB_SQLCommand mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabase initialisiert. Der Baustein FB_SQLCommand verwendet nur die Datenbankverbindung, die ihm über die CreateCmd-Methode zugewiesen wurde. Mehrere Instanzen vom Baustein FB_SQLCommand können mit derselben Datenbankverbindung initialisiert werden.

Die Initialisierung des Bausteins FB_SQLCommand ist im selben Zyklus abgeschlossen. Somit muss weder das Busy-Flag des Bausteins noch der Methodenrückgabewert der CreateCmd-Methode überprüft werden.

Syntax

```
METHOD CreateCmd : BOOL
VAR_INPUT
    pSQLCommand: POINTER TO FB_SQLCommand;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pSQLCommand	POINTER TO FB_SQLCommand	Liefert eine neue Instanz des FB_SQLCommand-Bausteins zurück.

Rückgabewert

Name	Typ	Beschreibung
CreateCmd	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR
```

```
// create a command reference
IF fbSqlDatabase.CreateCmd(ADR(fbSqlCommand)) THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLCommandEvt [▶ 200] zur Ausführung verwendet werden.

6.1.4.3.2.3 CreateSP

Mit dieser Methode wird eine Instanz des Bausteins FB_SQLStoredProcedure mit der bereits geöffneten Datenbankverbindung des Bausteins FB_SQLDatabase initialisiert. Der Baustein FB_SQLStoredProcedure verwendet nur die Datenbankverbindung, die ihm über die CreateCmd-Methode zugewiesen wurde. Mehrere Instanzen vom Baustein FB_SQLStoredProcedure können mit derselben Datenbankverbindung initialisiert werden.

Die Initialisierung des Bausteins FB_SQLStoredProcedure kann einige Zyklen in Anspruch nehmen. Das Busy-Flag des Bausteins oder der Methodenrückgabewert der CreateCmd-Methode muss überprüft werden bevor der Baustein einsatzbereit ist.

Syntax

```
METHOD CreateSP : BOOL
VAR_INPUT
    sProcedureName: T_MaxString;
    pParameterInfo: POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter;
    cbParameterInfo: UDINT;
    pSQLProcedure: POINTER TO FB_SQLStoredProcedure;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sProcedureName	T_MaxString	Gibt den Namen der Prozedur an, welche ausgeführt werden soll.
pParameterInfo	POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter	Pointer-Adresse zu der Parameter-Infoliste.
cbParameterInfo	UDINT	Gibt die Länge der Parameter-Infoliste an.
pSQLProcedure	POINTER TO FB_SQLStoredProcedure	Liefert eine neue Instanz des Bausteins FB_SQLStoredProcedure zurück.

Rückgabewert

Name	Typ	Beschreibung
CreateSP	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase   : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
    ParaInfo        : ST_SQLSPParameter;
END_VAR
```

```
ParaInfo.sParameterName     := '@Customer_ID';
ParaInfo.eParameterType     := E_SPParameterType.Input;
ParaInfo.eParameterDataType := E_ColumnType.BigInt;
ParaInfo.nParameterSize     := 8;

IF fbSQLDatabase.CreateSP('dbo.SP_GetCustomerPositions', ADR(ParaInfo), SIZEOF(ParaInfo), ADR(fbSQLS
toredProcedure)) THEN
    IF fbSQLDatabase.bError THEN
        nState:=255;
    ELSE
        nState:= nState+1;
    END_IF
END_IF
```

Darauffolgend kann der FB_SQLStoredProcedureEvt [▶ 205] verwendet werden, um die gespeicherte Prozedur auszuführen.

6.1.4.3.2.4 Disconnect

Diese Methode schließt die Verbindung zur Datenbank, welche von dieser Bausteininstanz geöffnet wurde.

Syntax

```
METHOD Disconnect : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Disconnect	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR
```

```
// disconnect from database
IF fbSqlDatabase.Disconnect() THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

6.1.4.3.3 FB_SQLCommand

Funktionsbaustein zum Ausführen von SQL-Kommandos. Zuvor muss er mit dem Baustein FB_SQLDatabase initialisiert werden.

Syntax

Definition:

```
FUNCTION_BLOCK FB_SQLCommand
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Execute [▶ 288]	Lokal	Sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank.
ExecuteDataReturn [▶ 289]	Lokal	Sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom Baustein FB_SQLResult übergeben werden.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.3.3.1 Execute

Diese Methode sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pSQLCmd	POINTER TO BYTE	Gibt die Pointer-Adresse einer String-Variablen mit dem auszuführenden SQL-Befehl an.
cbSQLCmd	UDINT	Gibt die Länge des SQL-Befehls an, der ausgeführt werden soll.

Rückgabewert

Name	Typ	Beschreibung
Execute	POINTER TO BYTE	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Nutzt das vom FB_SQLDatabaseEvt.CreateCmd() [▶ 196] erstellte Kommando.

```
VAR
    fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
    tcMessage    : I_TcMessage;
END_VAR
```

```
// you can generate this with the SQL Query Editor
sCmd := 'INSERT INTO myTable_Double ( Timestamp, Name, Value) VALUES ( $'2018-01-31 14:59:27$', $'Te
mperature$', 21.3)';

// call sql command
IF fbSQLCommand.Execute(ADR(sCmd), SIZEOF(sCmd)) THEN
    IF fbSQLCommand.bError THEN
        tcMessage := fbSQLCommand.ipTcResult;
        nState := 255;
    ELSE
```

```
        nState := nState+1;
    END_IF
END_IF
```

6.1.4.3.3.2 ExecuteDataReturn

Diese Methode sendet das angegebene SQL-Kommando über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom Baustein FB_SQLResult übergeben werden.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
    pSQLDBResult: POINTER TO FB_SQLResult;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pSQLCmd	POINTER TO BYTE	Gibt die Pointer-Adresse einer String-Variablen mit dem auszuführenden SQL-Befehl an.
cbSQLCmd	UDINT	Gibt die Länge des SQL-Befehls an, der ausgeführt werden soll.
pSQLDBResult	POINTER TO FB_SQLResult [▶ 289]	Gibt eine neue Instanz des Bausteins FB_SQLResult zurück.

Rückgabewert

Name	Typ	Beschreibung
ExecuteDataReturn	POINTER TO BYTE	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

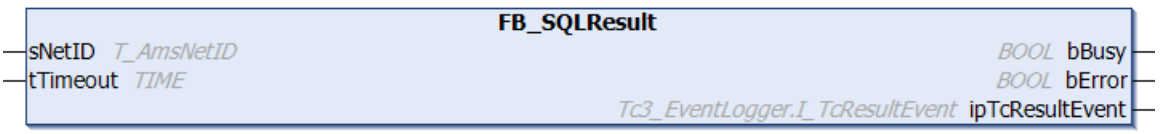
Nutzt das vom FB_SQLDatabaseEvt.CreateCmd() [▶ 196] erstellte Kommando.

```
VAR
    fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
    tcMessage    : I_TcMessage;
END_VAR
```

```
// you can generate this with the SQL Query Editor
sCmd := 'SELECT ID, Timestamp, Name, Value FROM myTable_Double';

// call sql command
IF fbSQLCommand.ExecuteDataReturn(ADR(sCmd), SIZEOF(sCmd), ADR(fbSqlResult)) THEN
    IF fbSQLCommand.bError THEN
        nState := 255;
    ELSE
         tcMessage := fbSQLCommand.ipTcResult;
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLResultEvt [▶ 203] genutzt werden, um die Daten auszulesen.

6.1.4.3.4 FB_SQLResult



Der Baustein dient zum Auslesen der gepufferten Datensätze.

Syntax

Definition:

```
FUNCTION_BLOCK FB_SQLResult
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Read [▶ 290]	Lokal	Liest eine vorgegebene Anzahl von Datensätzen aus den im TwinCAT Database Server gepufferten Ergebnisdaten aus.
Release [▶ 291]	Lokal	Gibt vom TwinCAT Database Server gepufferte Daten wieder frei.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.3.4.1 Read

Diese Methode liest eine vorgegebene Anzahl von Datensätzen aus den im TwinCAT Database Server gepufferten Ergebnisdaten aus.

Syntax

```
METHOD Read : BOOL
VAR_INPUT
    nStartIndex: UDINT := 0;
    nRecordCount: UDINT := 1;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    bWithVerifying: BOOL := FALSE;
    bDataRelease: BOOL := TRUE;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
nStartIndex	UDINT	Gibt den Index des ersten zu lesenden Datensatzes an.
nRecordCount	UDINT	Gibt die Anzahl der zu lesenden Datensätzen an.
pData	POINTER TO BYTE	Adresse des Strukturarrays, in das die Datensätze geschrieben werden sollen.
cbData	UDINT	Gibt die Größe des Strukturarrays in Byte an.
bWithVerifying	BOOL	Rückgabedaten werden mit pData Strukturarray verglichen und ggf. angepasst.
bDataRelease	BOOL	Gibt die gepufferten Daten wieder frei.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

```
VAR
    fbSqlResult : FB_SQLResultEvt(sNetID:='', tTimeout := T#5S);
    aReadStruct : ARRAY[1..5] OF ST_StandardRecord;
END_VAR
```

```
// get values from internal tc db srv storage
IF fbSqlResult.Read(2, 3, ADR(aReadStruct), SIZEOF(aReadStruct), FALSE, TRUE) THEN
    IF fbSqlResult.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Ergebnis in der SPS:

Expression	Type	Value
aReadStruct	ARRAY [1..5] OF ST...	
aReadStruct[1]	ST_StandardRecord	
nID	LINT	9
dtTimestamp	DATE_AND_TIME	DT#2018-1-31-15:4:59
sName	STRING(80)	'Temperature'
rValue	LREAL	21.3
aReadStruct[2]	ST_StandardRecord	
nID	LINT	10
dtTimestamp	DATE_AND_TIME	DT#2018-1-31-15:5:59
sName	STRING(80)	'Temperature'
rValue	LREAL	21.2

6.1.4.3.4.2 Release

Mit dieser Methode können vom TwinCAT Database Server gepufferte Daten wieder freigegeben werden.

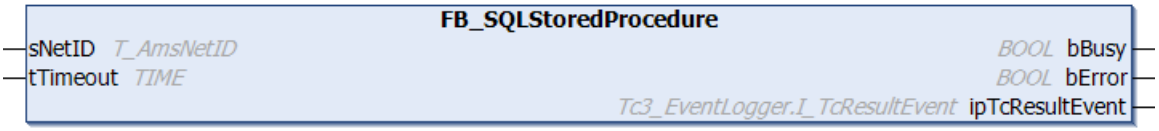
Syntax

```
METHOD Release : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Release	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

6.1.4.3.5 FB_SQLStoredProcedure



Funktionsbaustein zum Ausführen von gespeicherten Prozeduren (Stored Procedures) der Datenbank. Zuvor muss dieser mit dem Baustein "FB_SQLDatabase" initialisiert werden.

Syntax

Definition:

```
FUNCTION BLOCK FB_SQLStoredProcedure
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

Eingänge

Name	Typ	Beschreibung
sNetID	T_AmsNetID	AMS-Netzwerkkennung des Zielgerätes, an das der ADS-Befehl gerichtet ist.
tTimeout	TIME	Gibt die Zeit bis zum Abbruch der Funktion an.

Ausgänge

Name	Typ	Beschreibung
bBusy	BOOL	TRUE, sobald eine Methode des Bausteins aktiv ist.
bError	BOOL	TRUE, sobald ein Fehler eintritt.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Ergebnis-Interface, welches den Rückgabewert detailliert angibt.

Methoden

Name	Definitionsort	Beschreibung
Execute [▶ 293]	Lokal	Sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank.
ExecuteDataReturn [▶ 294]	Lokal	Sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom FB_SQLDatabase-Baustein geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom FB_SQLResult-Baustein übergeben werden.
Release [▶ 294]	Lokal	Gibt die Parameterinformationen der gespeicherten Prozedur (Stored Procedure), die bei der Initialisierung übergeben wurden, wieder frei.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.1 Build 4020.10	PC oder CX (x86)	Tc3_Database

6.1.4.3.5.1 Execute

Diese Methode sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom Baustein FB_SQLDatabase geöffnete Datenbankverbindung an die Datenbank.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pParameterStrc	POINTER TO BYTE	Pointer-Adresse zur Parameterstruktur, die der Prozedur übergeben wird.
cbParameterStrc	UDINT	Länge der Parameterstruktur

Rückgabewert

Name	Typ	Beschreibung
Execute	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Nutzt die zuvor mit FB_SQLDatabaseEvt.CreateSP() [▶ 196] erstellte Stored Procedure

```
VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID:='', tTimeout := T#5S);
    Customer_ID          : LINT;
    tcMessage            : I_TcMessage;
END_VAR
```

```
IF fbSQLStoredProcedure.Execute(pParameterStrc := ADR(Customer_ID) , cbParameterStrc:= SIZEOF(Custom
er_ID)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
```

```
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

6.1.4.3.5.2 ExecuteDataReturn

Diese Methode sendet den Aufruf der angegebenen gespeicherten Prozedur (Stored Procedure) über die bereits vom FB_SQLDatabase-Baustein geöffnete Datenbankverbindung an die Datenbank. Um die zurückgelieferten Datensätze zu lesen, kann eine Instanz vom FB_SQLResult-Baustein übergeben werden.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
    pSQLDBResult: POINTER TO FB_SQLDBResult;
END_VAR
```

Eingänge

Name	Typ	Beschreibung
pParameterStrc	POINTER TO BYTE	Pointer-Adresse zur Parameterstruktur, die der Prozedur übergeben wird.
cbParameterStrc	UDINT	Länge der Parameterstruktur
pSQLDBResult	POINTER TO FB_SQLDBResult	Gibt eine neue Instanz des Bausteins FB_SQLDBResult zurück.

Rückgabewert

Name	Typ	Beschreibung
Read	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

Beispiel

Nutzt die zuvor mit FB_SQLDatabaseEvt.CreateSP() [▶ 196] erstellte Stored Procedure

```
VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID:='', tTimeout := T#5S);
    Customer_ID          : LINT;
    tcMessage            : I_TcMessage;
END_VAR
```

```
IF fbSQLStoredProcedure.ExecuteDataReturn(pParameterStrc := ADR(Customer_ID), cbParameterStrc:= SIZE
OF(Customer_ID), pSQLDBResult := ADR(fbSqlResult)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLResultEvt [▶ 203] genutzt werden, um die Daten auszulesen.

6.1.4.3.5.3 Release

Diese Methode gibt die Parameterinformationen der gespeicherten Prozedur (Stored Procedure), die bei der Initialisierung übergeben wurden, wieder frei.

Syntax

```
METHOD Release : BOOL
```

Rückgabewert

Name	Typ	Beschreibung
Release	BOOL	Zeigt den Status der Methode. Liefert TRUE, sobald die Methodenausführung beendet ist, also auch im Fehlerfall.

6.2 Tc2_Database

Übersicht

Die Tc2_Database-Bibliothek beinhaltet Funktionsblöcke zum Steuern und Konfigurieren des TwinCAT 3 Database Servers.

Funktionsblöcke

Name	Beschreibung
FB_GetStateTcDatabase [▶ 297]	Ruft Statusinformationen ab.
FB_DBConnectionAdd [▶ 299]	Fügt der XML-Konfigurationsdatei Datenbankanbindungen an.
FB_DBAuthentificationAdd [▶ 319]	Fügt der XML-Konfigurationsdatei Authentifizierungsinformationen der jeweiligen Datenbankanbindung an.
FB_DBOdbcConnectionAdd [▶ 300]	Fügt der XML-Konfigurationsdatei eine ODBC-Datenbankverbindung an.
FB_AdsDeviceConnectionAdd [▶ 302]	Fügt der XML-Konfigurationsdatei ein ADS-Gerät an.
FB_DBReloadConfig [▶ 298]	Lädt die XML-Konfigurationsdatei neu.
FB_GetDBXMLConfig [▶ 303]	Liest alle Datenbankkonfigurationen aus der XML-Konfigurationsdatei aus.
FB_GetAdsDevXMLConfig [▶ 303]	Liest alle ADS-Gerätekonfigurationen aus der XML-Konfigurationsdatei aus.
FB_DBConnectionOpen [▶ 304]	Öffnet eine Verbindung zu einer Datenbank.
FB_DBConnectionClose [▶ 305]	Schließt eine Verbindung zu einer Datenbank.
FB_DBCreate [▶ 306]	Erstellt eine neue Datenbank.
FB_DBTableCreate [▶ 307]	Erstellt eine Tabelle mit beliebiger Tabellenstruktur.
FB_DBRead [▶ 309]	Liest einen Wert aus der Datenbank aus.
FB_DBWrite [▶ 310]	Schreibt einen Variablenwerte mit Timestamp in eine Datenbank.
FB_DBCyclicRdWrt [▶ 308]	Startet bzw. stoppt das Loggen\Schreiben der Variablen.
FB_DBRecordSelect [▶ 321]	Liest einen Datensatz aus einer Tabelle aus.
FB_DBRecordSelect_EX [▶ 323]	Liest einen Datensatz aus einer Tabelle aus. (Befehlslänge <= 10000 Zeichen)
FB_DBRecordArraySelect [▶ 314]	Liest mehrere Datensätze aus einer Tabelle aus.
FB_DBRecordInsert [▶ 320]	Erstellt einen neuen Datensatz.
FB_DBRecordInsert_EX [▶ 313]	Erstellt einen neuen Datensatz. (Befehlslänge <= 10000 Zeichen)
FB_DBRecordDelete [▶ 312]	Löscht einen Datensatz aus einer Tabelle.
FB_DBStoredProcedures [▶ 316]	Führt eine gespeicherte Prozedur (Stored Procedure) aus.
FB_DBStoredProceduresRecordReturn [▶ 324]	Führt eine gespeicherte Prozedur (Stored Procedure) aus und gibt einen Datensatz zurück.
FB_DBStoredProceduresRecordArray [▶ 317]	Führt eine gespeicherte Prozedur (Stored Procedure) aus und gibt mehrere Datensätze zurück.

Datentypen

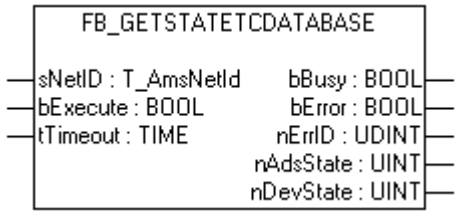
Name
ST_DBColumnCfg [▶ 325]
ST_DBXMLCfg [▶ 325]
ST_ADSDevXMLCfg [▶ 326]
ST_DBSQLError [▶ 326]
ST_DBParameter [▶ 326]
E_DbColumnTypes [▶ 327]
E_DBTypes [▶ 328]
E_DBValueType [▶ 328]
E_DBWriteModes [▶ 328]
E_DBParameterTypes [▶ 328]

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1 Funktionsbausteine

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

bExecute: Mit der steigenden Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Timeout-Zeit 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

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 v3.0.0	PC oder CX (x86)	Tc2_Database

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

bExecute: Mit steigender 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 ist, wird kein neues Kommando angenommen.

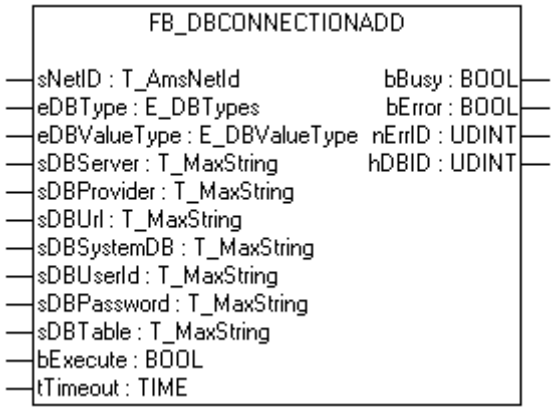
bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

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

eDBType: Gibt den Typ 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 Passwort an.

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

bExecute: Mit steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

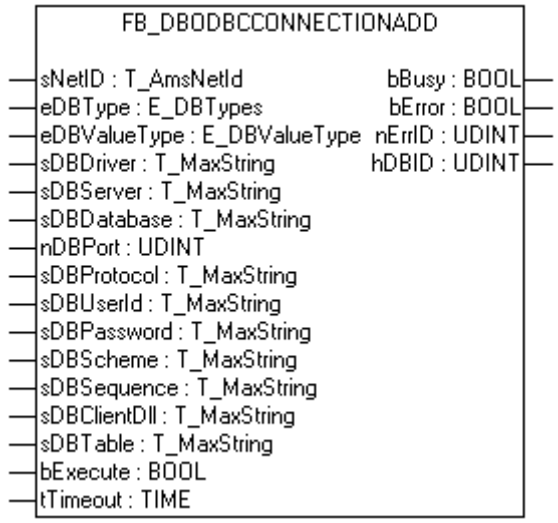
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

hDBID: Liefert die ID der Datenbank zurück.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.4 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: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

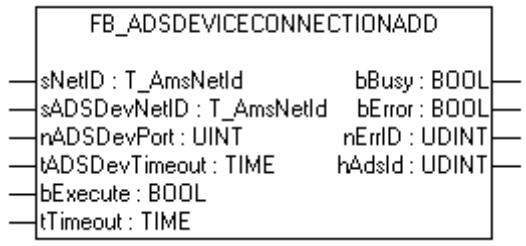
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

hDBID: Liefert die ID der Datenbank zurück

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

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

sADSDevNetID: 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 steigender 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

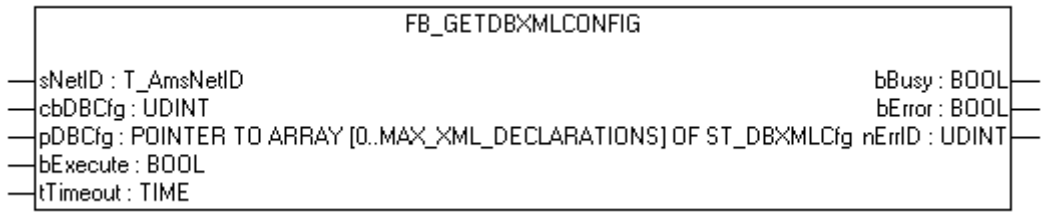
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

hAdsId: Gibt die ID des ADS-Gerätes zurück.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.6 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: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

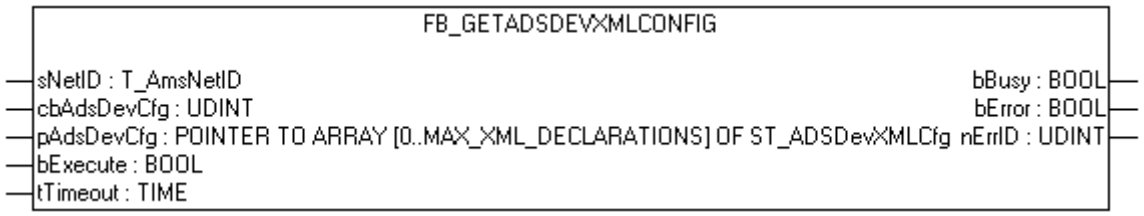
bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.7 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: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

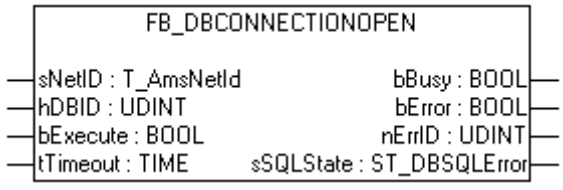
bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Databas

6.2.1.8 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: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

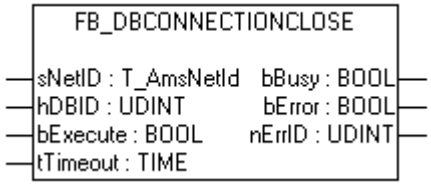
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv_Error_Codes [▶ 406].

sSQLState : Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.9 FB_DBConnectionClose



Mit dem Funktionsbaustein FB_DBConnectionClose 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: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

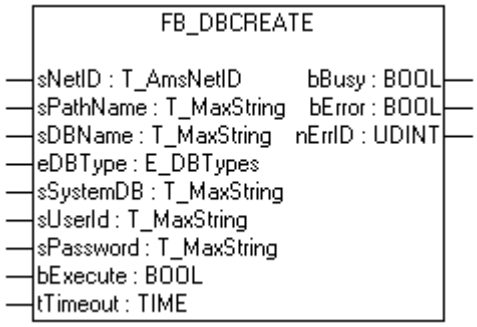
bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.10 FB_DBCreate



Mit dem Funktionsbaustein FB_DBCreate können Datenbanken angelegt werden.

Folgende Datenbanken können mit diesem Baustein erstellt werden: MS SQL Datenbanken, MS SQL Compact Datenbanken, MS Access Datenbanken und XML Datenbanken

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: 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 für Access Datenbanken. Beinhaltet den Pfad zu der MDW Datei.

sUserID: Nutzername für die entsprechende Registrierung

sPassword: Zugehöriges Passwort

bExecute: Mit steigender 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

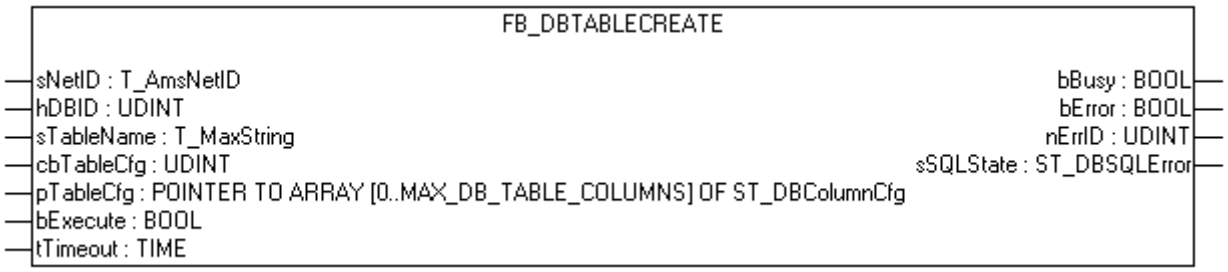
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

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

hDBID: 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 Tablestructarrays an. In diesem Array werden die einzelnen Spalten beschrieben.

bExecute: Mit steigender 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

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

bExecute: Mit steigender Flanke wird der Lese-/Schreibzyklus gestartet und mit fallender 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

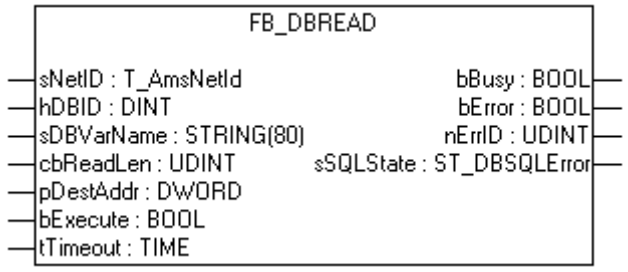
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.13 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 : POINTER TO BYTE;
    bExecute  : BOOL;
    tTimeout  : TIME;
END_VAR
```

sNetID: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

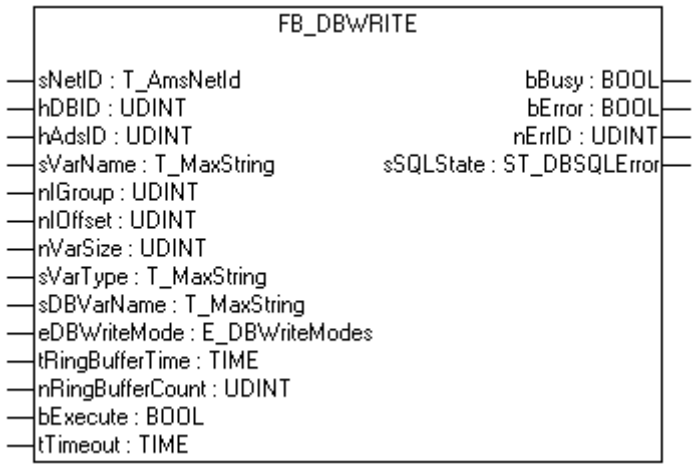
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.14 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" [▶ 131]). 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: String, der die AMS-Netzwerkkennung des Zielgerätes enthält, an das der ADS-Befehl gerichtet wird.

hDBID: ID der zu verwendenden Datenbank.

hAdsID: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv_Error_Codes [▶ 406].

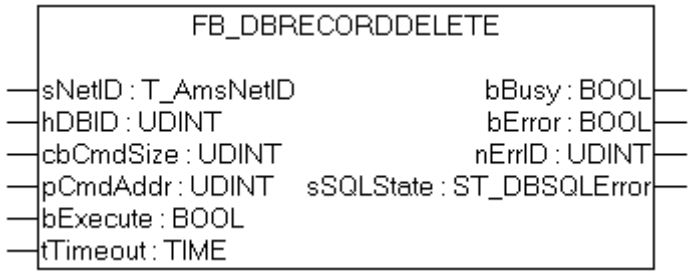
sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps

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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.15 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 : POINTER TO BYTE;
    bExecute : BOOL;
    tTimeout : TIME;
END_VAR
```

sNetID: 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 einer steigenden Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

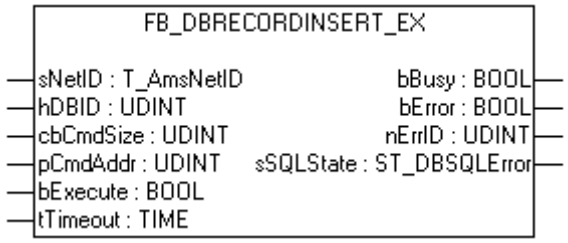
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.16 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 : POINTER TO BYTE;
    bExecute : BOOL;
    tTimeout : TIME;
END_VAR
```

sNetID: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

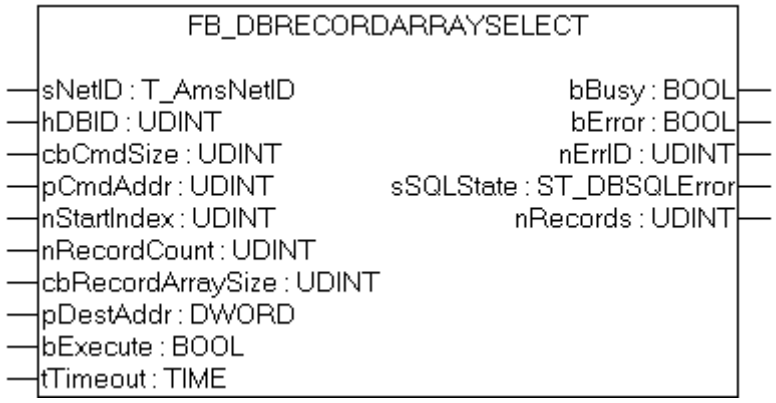
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.17 FB_DBRecordArraySelect



Mit dem Funktionsbaustein FB_DBRecordArraySelect können mehrere 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;
    nStartIndex      : UDINT;
    nRecordCount     : UDINT;
    cbRecordArraySize: UDINT;
    pDestAddr        : POINTER TO BYTE;
    bExecute         : BOOL;
    tTimeout         : TIME;
END_VAR
```

sNetID: 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 sollen.

bExecute: Mit steigender Flanke wird das Kommando ausgeführt.

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

VAR_OUTPUT

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

ST_DBSQLError [▶ 326]

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

bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL-Fehlercode des entsprechenden Datenbanktyps.

nRecords: Liefert die Anzahl der Datensätze.

Beispiel in ST

Da die Tabelle, aus der Datensätze gelesen werden sollen, die nachfolgende Struktur besitzt, muss eine SPS-Struktur erstellt werden, die einen vergleichbaren Aufbau besitzt.

Tabelle:

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

Struktur:

```
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 aufgrund des Byte-Alignements 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
```

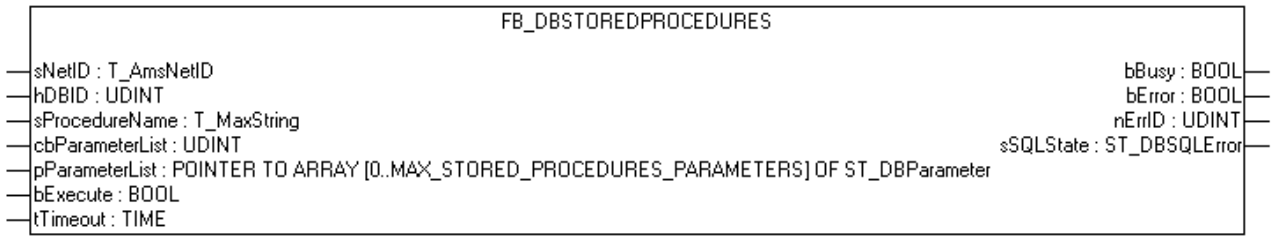
SPS-Programm

```
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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.18 FB_DBStoredProcedures



Mit dem Funktionsbaustein FB_DBStoredProcedures können gespeichert Prozeduren (Stored Procedures) aufgerufen werden. Sie können Parameter 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: 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 Prozedur an, welche ausgeführt werden soll

cbParameterList: Gibt die Länge der Parameterliste an.

pParameterList: Enthält die Adresse der Parameterliste

bExecute: Mit einer steigenden Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

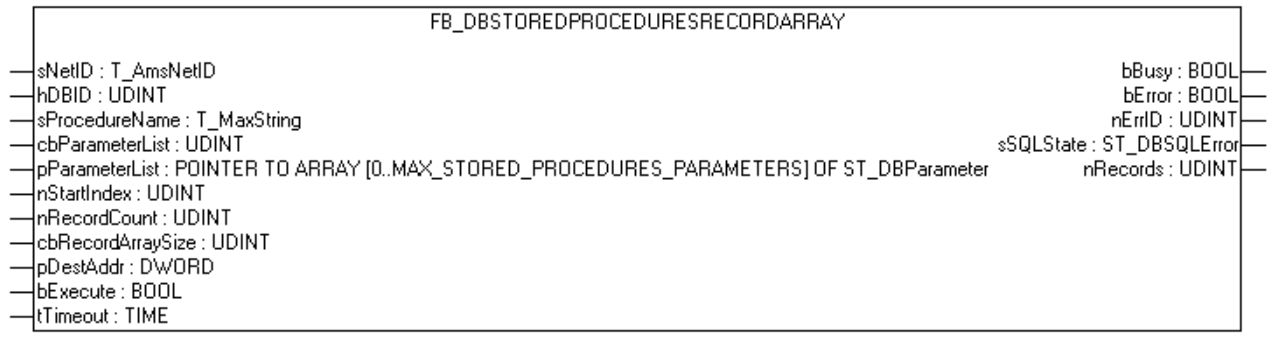
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.19 FB_DBStoredProceduresRecordArray



Mit dem Funktionsbaustein FB_DBStoredProceduresRecordArray können gespeichert Prozeduren (Stored Procedures) aufgerufen werden, die Datensätze zurückliefern. Im Gegensatz zum FB_DBStoredProceduresRecordReturn können mit diesem Baustein auch mehrere Datensätze mit einem Aufruf zurückgeliefert werden. Sie können Parameter 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         : POINTER TO BYTE;
 bExecute         : BOOL;
 tTimeout         : TIME                  := T#15s;
END_VAR
```

sNetID: 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 Prozedur 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 Datensätze an.

cbRecordArraySize: Gibt die Größe des Strukturarrays in Byte an.

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

bExecute: Mit steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 bzw. TcDatabaseSrv_Error_Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

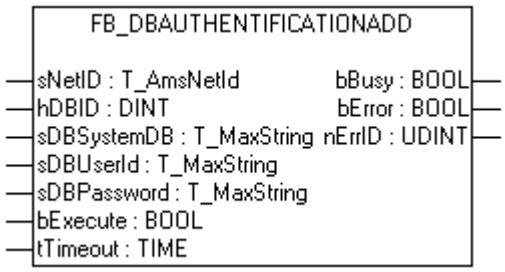
nRecords: Liefert die Anzahl der Datensätze.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.20 Veraltet

6.2.1.20.1 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: 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 Passwort an.

bExecute: Mit steigender 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.

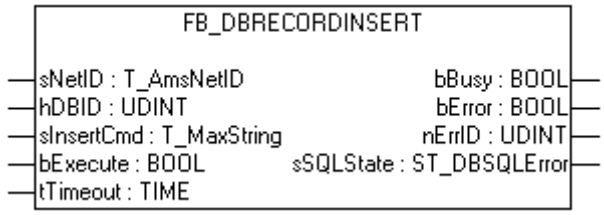
bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.20.2 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: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

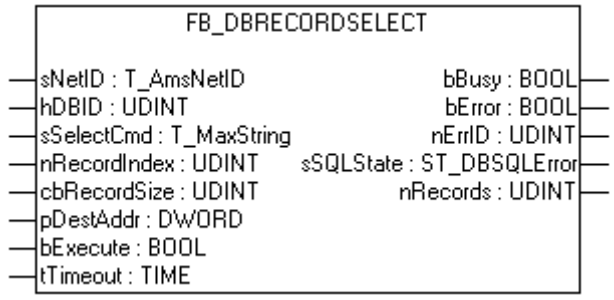
nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.20.3 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: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

nRecords: Liefert die Anzahl der Datensätze.

Beispiel in ST:

Da die Tabelle, aus der ein Datensatz gelesen werden soll, die nachfolgende Struktur besitzt, muss eine SPS-Struktur erstellt werden, die einen vergleichbaren Aufbau besitzt.

Tabelle:

Spaltenname	Datentyp
ID	Bigint
Timestamp	datetime
Name	Ntext
Value	Float

Struktur:

```
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 aufgrund 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
```

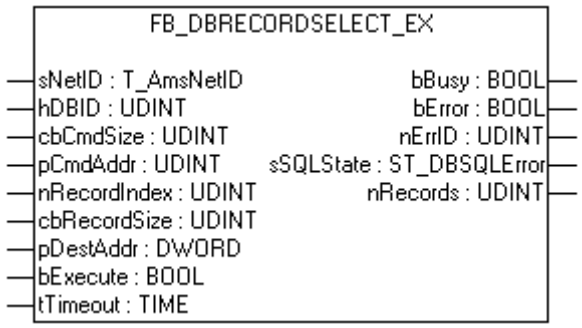
SPS-Programm

```
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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.20.4 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   : POINTER TO BYTE;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetID: 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv_Error_Codes [▶ 406].

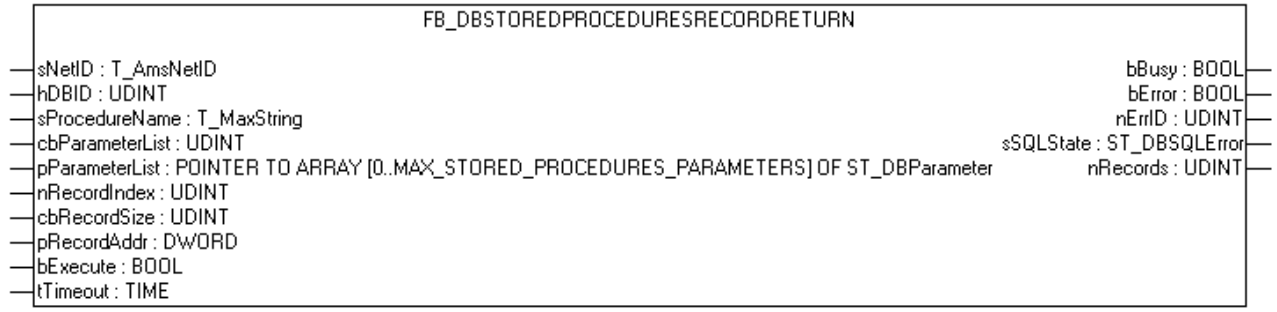
sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

nRecords: Liefert die Anzahl der Datensätze.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.1.20.5 FB_DBStoredProceduresRecordReturn



Mit dem Funktionsbaustein FB_DBStoredProceduresRecordReturn können gespeichert Prozeduren (Stored Procedures) aufgerufen werden die einen Datensatz zurückliefert. Sie können Parameter 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     : POINTER TO BYTE;
 bExecute        : BOOL;
 tTimeout        : TIME           := T#15s;
END_VAR
```

sNetID: 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 Prozedur 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 steigender Flanke wird das Kommando ausgeführt.

tTimeout: Gibt die Zeit bis zum Abbruch 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 ist, wird kein neues Kommando angenommen.

bError: Wird TRUE, sobald ein Fehler eintritt.

nErrID: Liefert bei einem gesetzten bError-Ausgang den ADS Error Code bzw. TcDatabaseSrv Error Codes [▶ 406].

sSQLState: Liefert den SQL - Fehlercode [▶ 326] des entsprechenden Datenbanktyps.

nRecords: Liefert die Anzahl der Datensätze.

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2 Datentypen

6.2.2.1 ST_DBColumnCfg

VAR_INPUT

```
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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.2 ST_DBXMLCfg

VAR_INPUT

```
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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.3 ST_ADSDevXMLCfg

VAR_INPUT

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

sAdsDevNetID: 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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.4 ST_DBSQLError

VAR_INPUT

```
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: Liefert einen datenbankspezifischen Fehlercode.

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.5 ST_DBParameter

VAR_INPUT

```
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 des Parameters an (E_DBColumnTypes [▶ 327]).

eParameterType: Gibt den Typ des Parameters an (E_DBParameterTypes [▶ 328]).

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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.6 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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.7 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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.8 E_DBValueType

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

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.9 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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.2.10 E_DBParameterTypes

```
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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.3 Globale Konstanten

6.2.3.1 Konstanten

```
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 v3.0.0	PC oder CX (x86)	Tc2_Database

6.2.3.2 Bibliotheksversion

Alle Bibliotheken haben eine bestimmte Version. Diese Version ist u. a. im SPS-Bibliotheks-Repository zu sehen. Eine globale Konstante enthält die Information über die Bibliotheksversion:

Global_Version

```
VAR_GLOBAL CONSTANT
stLibVersion_TC3_Database_Server : ST_LibVersion;
END_VAR
```

Um zu prüfen, ob Sie die richtige Version verwenden, benutzen Sie die Funktion F_CmpLibVersion (definiert in Tc2_System library).

Alle anderen Möglichkeiten Bibliotheksversionen zu vergleichen, die Sie von TwinCAT 2 kennen, sind veraltet!

7 Beispiele

7.1 Tc3_Database

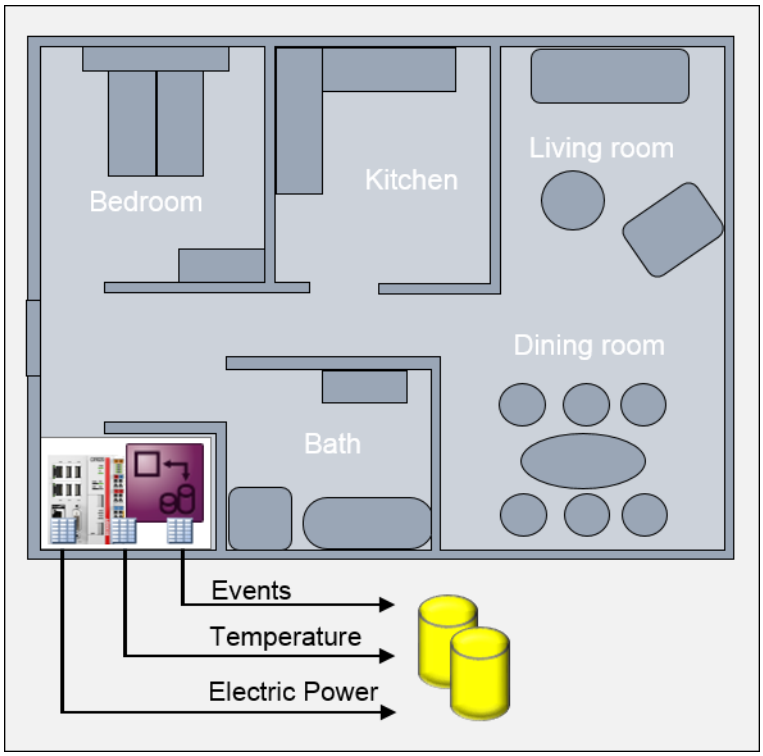
Auf den folgenden Seiten finden Sie Beispiele und Tipps für die Anwendung des TwinCAT Database Servers mit der Bibliothek für TwinCAT 3.

7.1.1 Szenario-Beispiele

Die einzelnen Beispiele sind jeweils einem Szenario zugewiesen, das auf eigene Anwendungsfälle adaptiert werden kann. Zusätzlich ist beschrieben, mit welchen Datenbanken die Beispiele kompatibel sind.

7.1.1.1 Home Automation

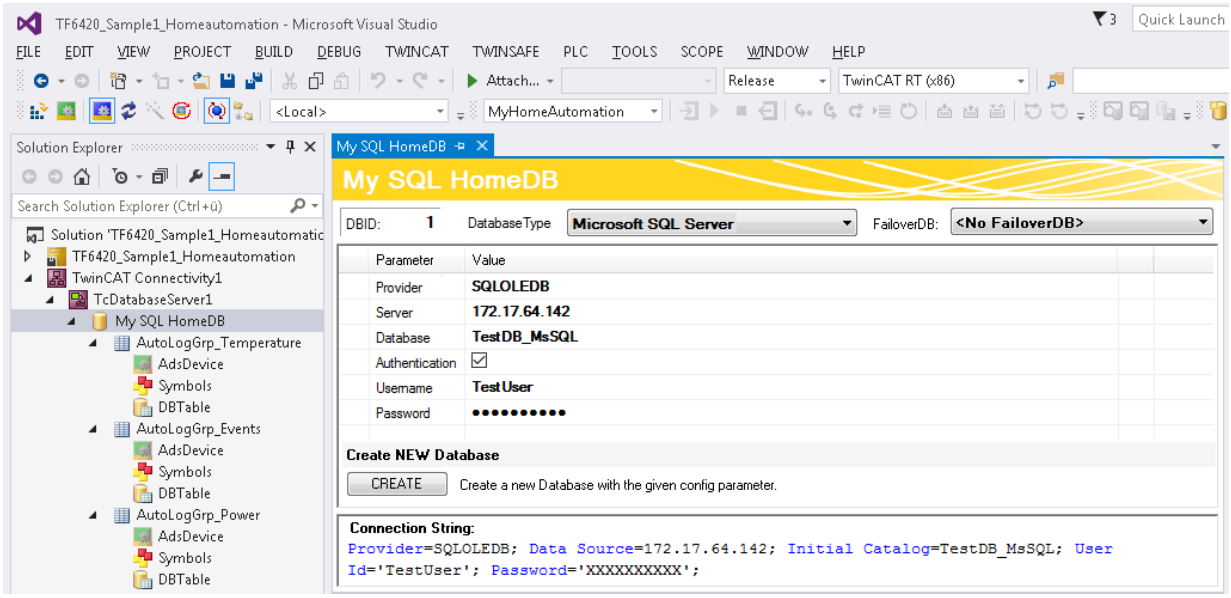
In diesem Szenario wird der Konfigurationsmodus beispielhaft im Bereich der Gebäudeautomatisierung gezeigt. Symbole werden in drei verschiedenen AutoLog-Gruppen ohne zusätzliche Programmierung, also durch reine Konfiguration, in eine MySQL Datenbank geschrieben. Raumtemperaturen werden im 5-Minuten-Zyklus in die Datenbank geloggt. Energiedaten werden im 1-Minuten-Takt gespeichert. Events wie „Lampe An“, „Lampe Aus“ werden gespeichert.



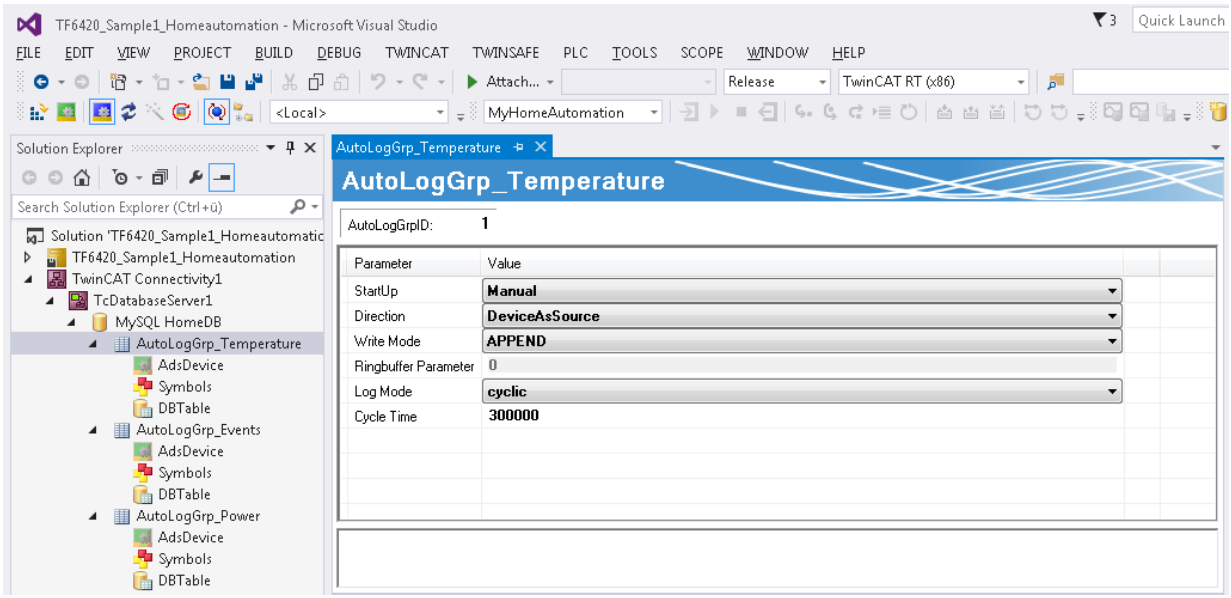
Kategorie	Configure Mode
Verwendete Datenbank	MySQL
Kompatible Datenbanken	Für alle unterstützten Datenbanktypen anwendbar
Verwendete SPS Funktionsbausteine	Keine
Verwendete SPS Bibliotheken	Tc3_Database, Tc2_BABasic
Download	https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3495185419.zip

✓ TwinCAT-Database-Server-Projekt wurde erstellt.

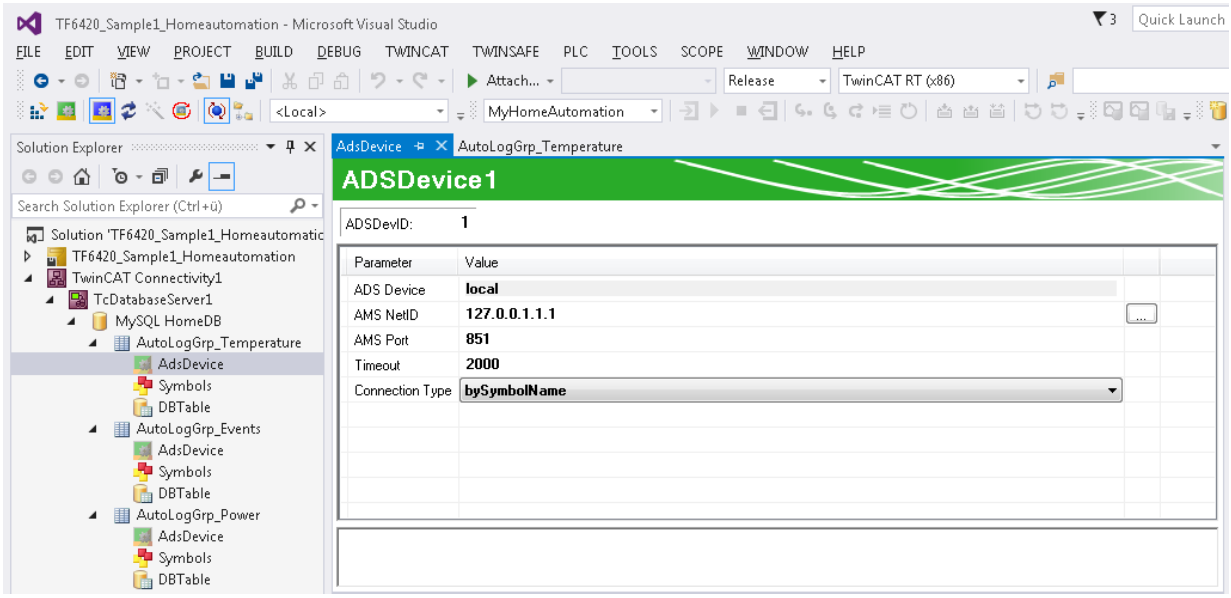
1. Zunächst wird die Datenbankverbindung hinzugefügt und konfiguriert.



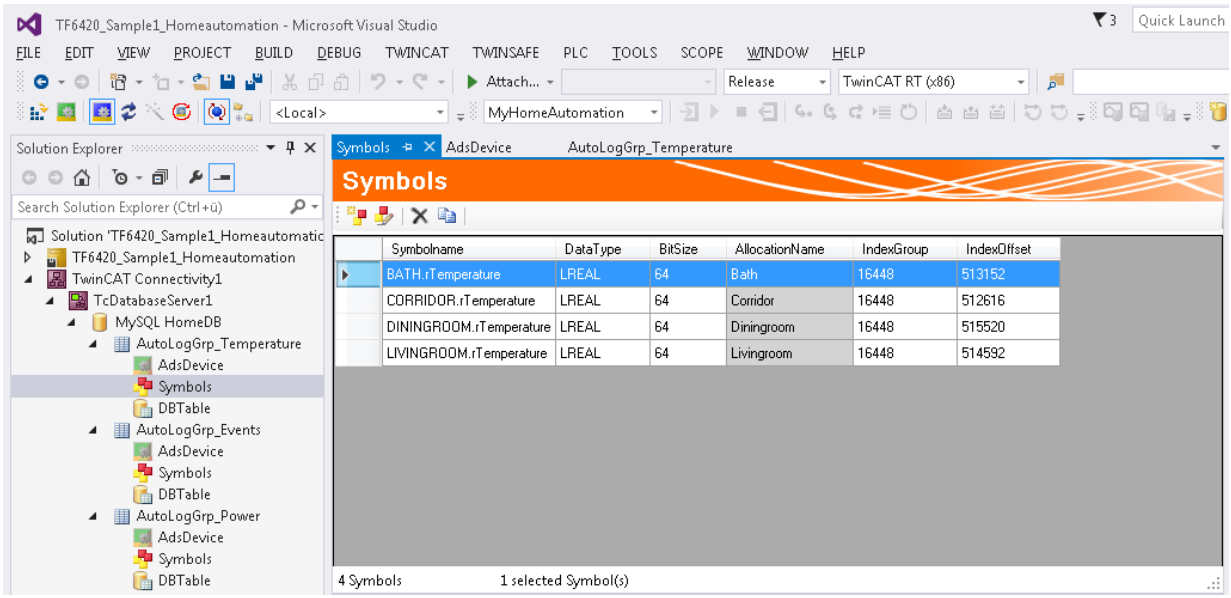
2. Da in drei verschiedene Tabellen geloggt werden soll, werden drei AutoLog-Gruppen an die Datenbankverbindung angehängt. Zunächst wird die Temperatur-AutoLog-Gruppe konfiguriert. Da alle 5 Minuten ein Wert in die Datenbank geschrieben werden soll, wird die CycleTime auf 300000 ms gesetzt.



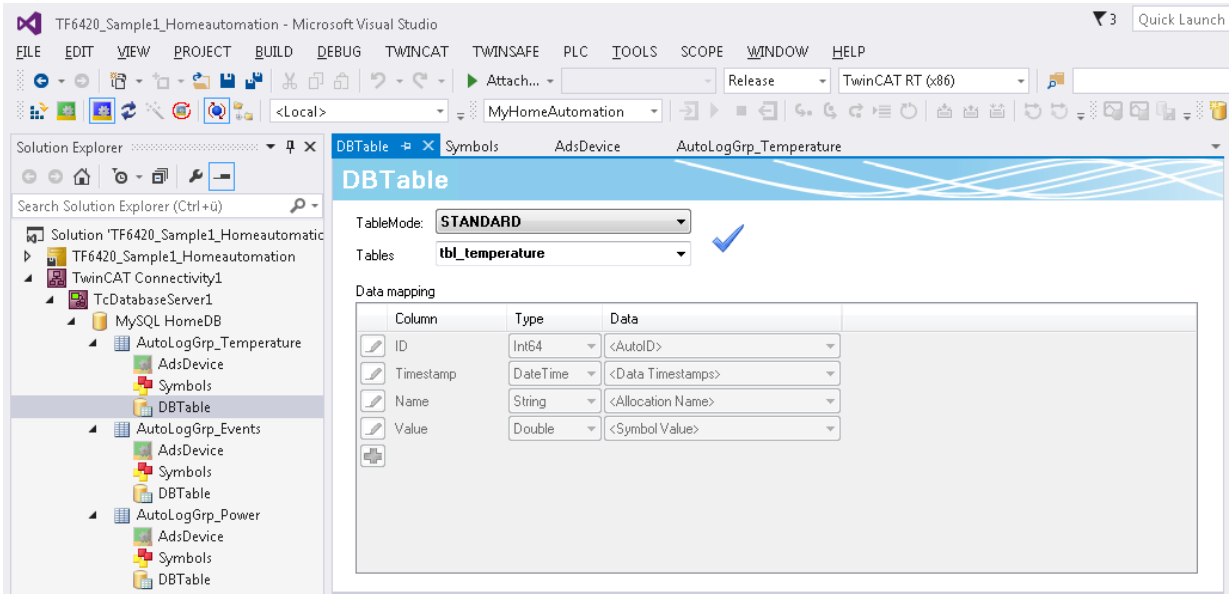
3. Danach wird das ADS-Gerät, von dem die ADS-Symbole ausgelesen werden sollen, eingestellt.



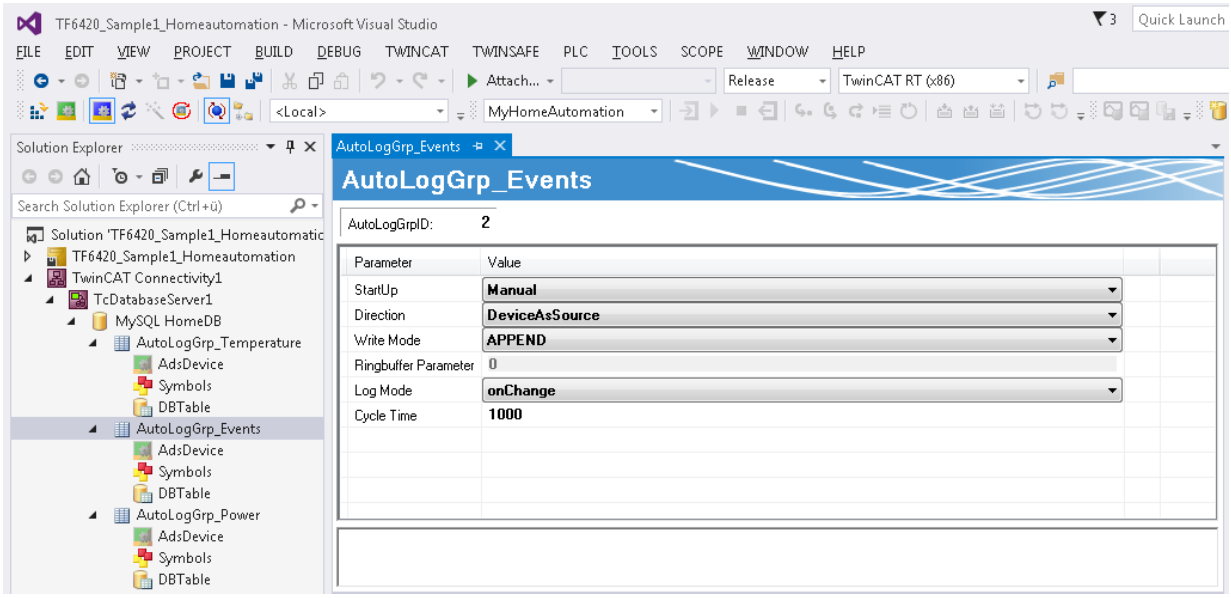
4. Die Symbole werden mithilfe des Target Browsers angelegt.



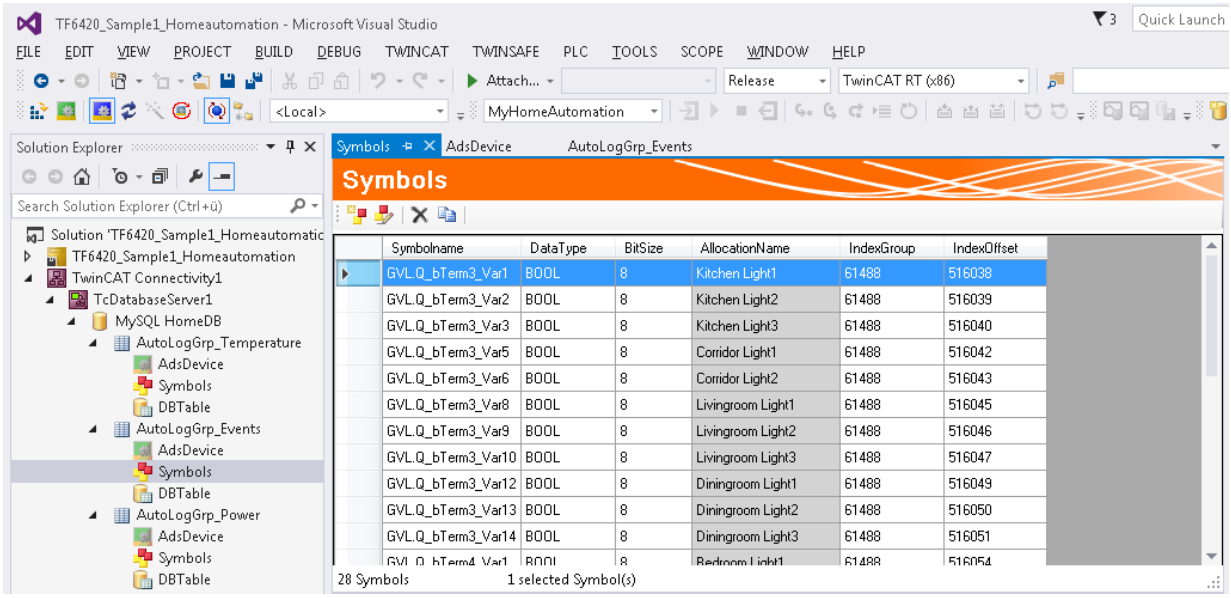
5. Nun kann die Tabelle gewählt werden, in die die Temperaturwerte geschrieben werden sollen. Die AutoLog-Gruppen unterstützen zwei TableModes [▶ 41]. Für die Temperaturwerte wird die Standard-Tabellenstruktur verwendet. Ob die ausgewählte Tabelle die richtige Struktur besitzt, wird durch einen Haken signalisiert.



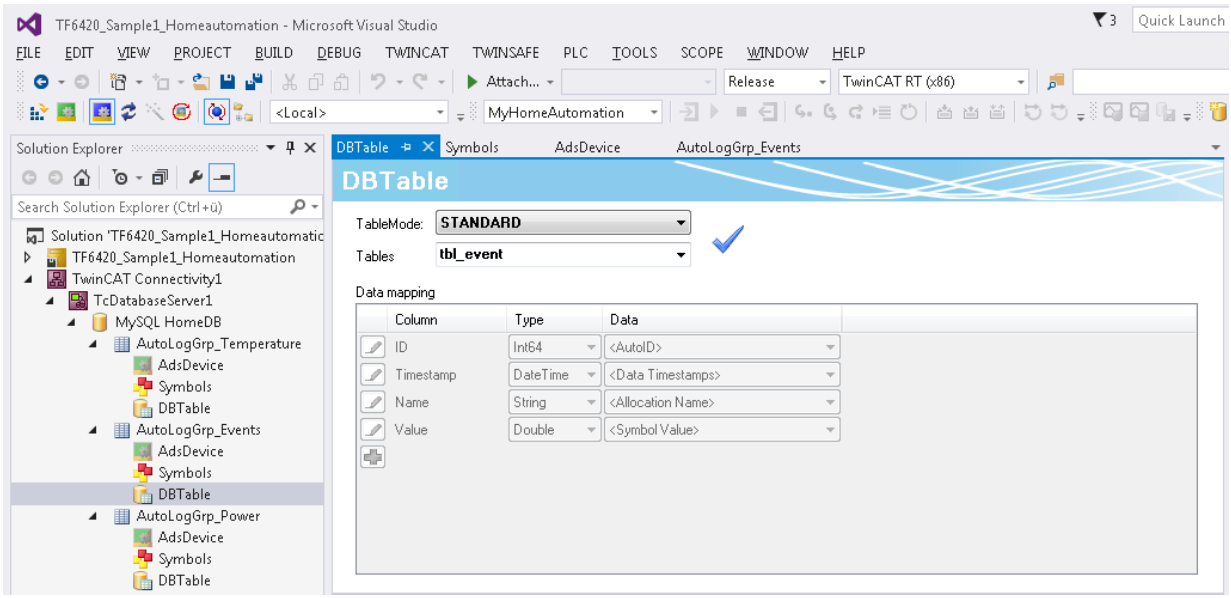
6. Dann wird die AutoLog-Gruppe für die Events konfiguriert. Für die Gruppe wird der LogMode auf „onChange“ gestellt und die Cycle Time auf 1000 ms. Dies bedeutet, dass die Symbole im 1-Sekunden-Takt überprüft werden und nur bei Wertänderung ein Eintrag in der Tabelle erzeugt wird.



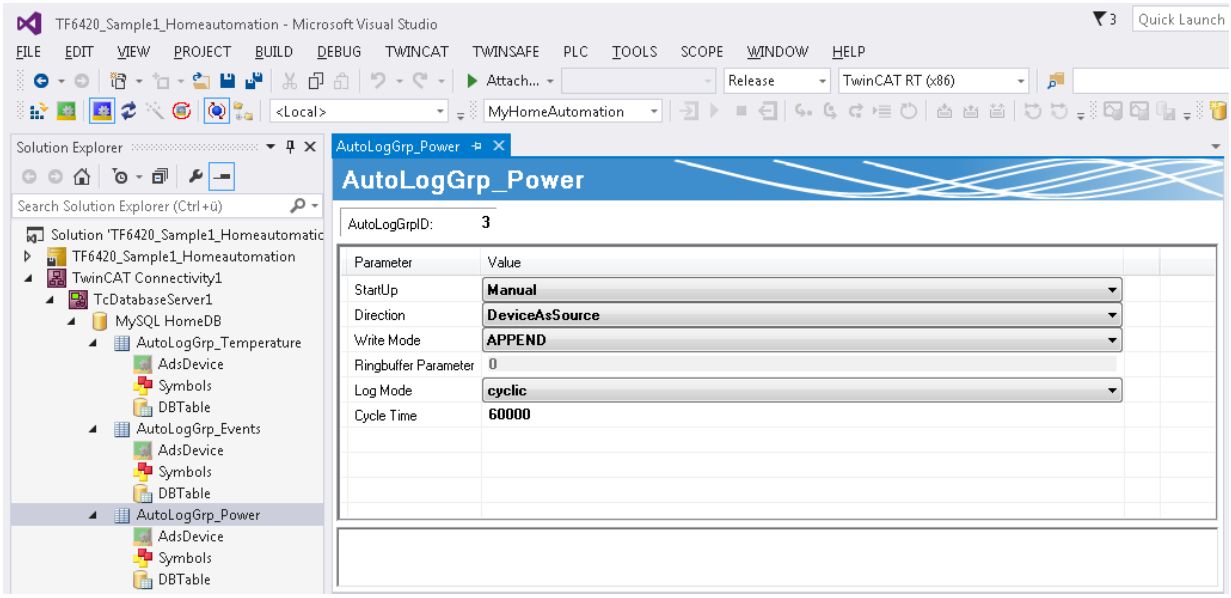
7. Anschließend werden wieder wie in Schritt 3 und 4 das ADS-Gerät und die Symbole ausgewählt.



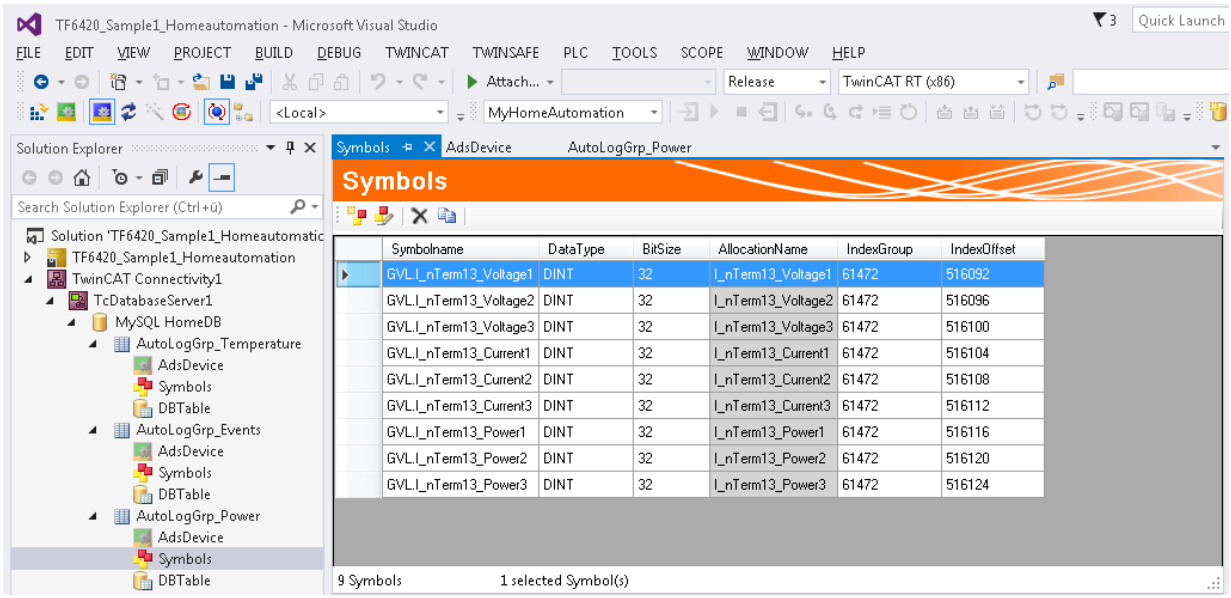
8. Auch die Events werden in einer Standard-Tabellenstruktur abgespeichert.



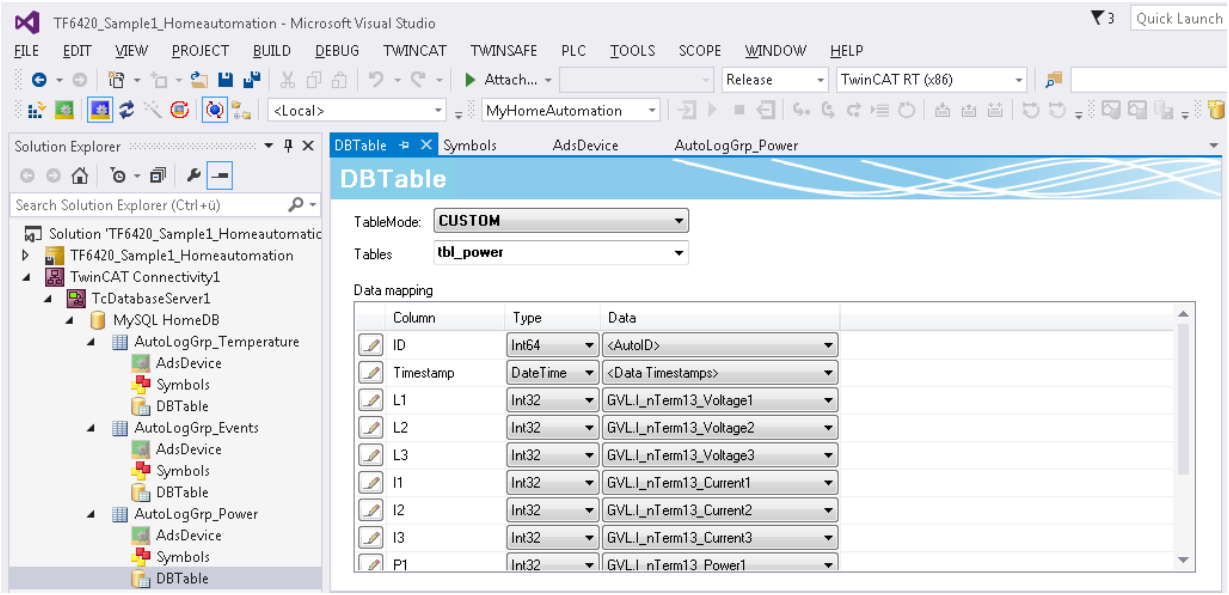
9. Zum Schluss werden die Energiedaten im 1-Minuten-Takt gespeichert.



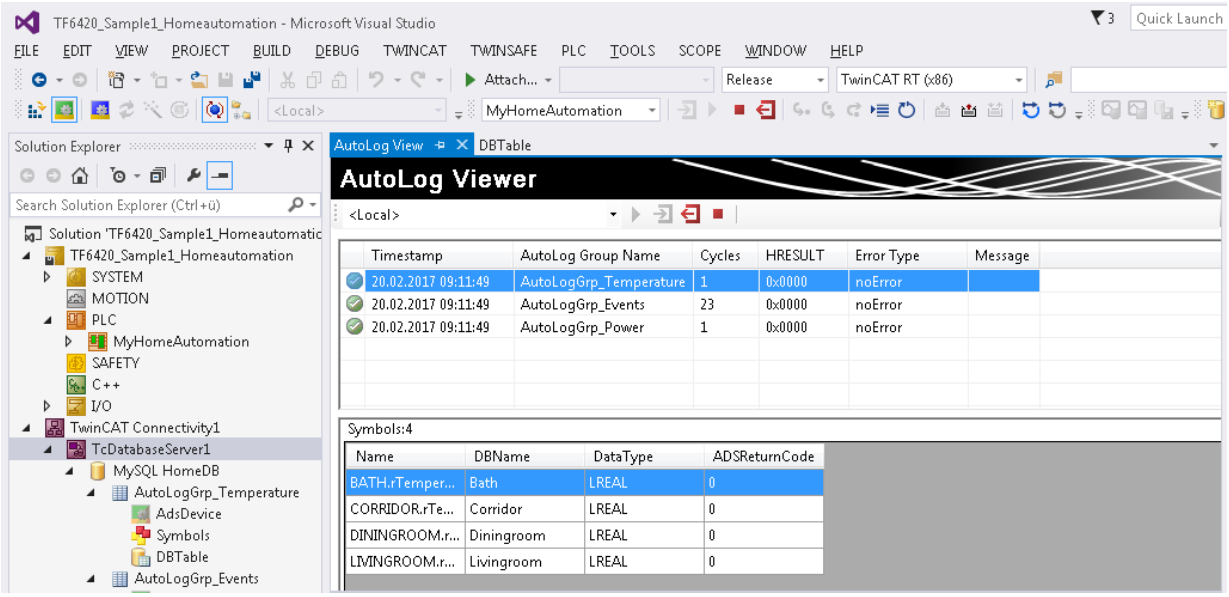
10. Auch für die Energiedaten werden das ADS-Gerät und die Symbole ausgewählt.



11. Die Energiedaten sollen in eine eigene „freie“ Tabellenstruktur gespeichert werden. Dafür werden die verschiedenen Symbole den Spalten zugeordnet.

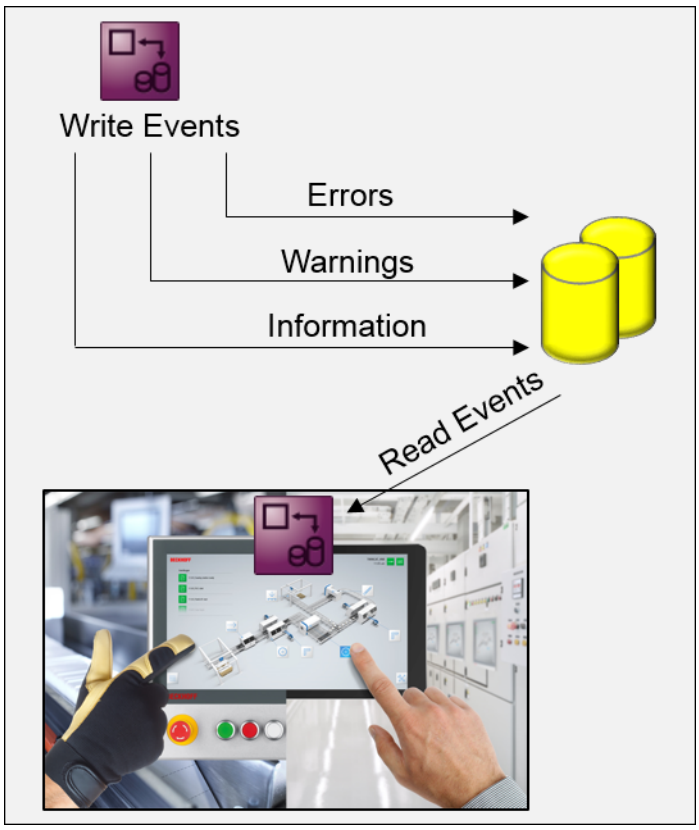


12. Nun kann mit dem AutoLog Viewer [▶ 50] das Loggen gestartet und überwacht werden.



7.1.1.2 Message Logger

In diesem Szenario wird der PLC Expert Mode beispielhaft für einen Message Logger in der SPS gezeigt. In dem Beispielprogramm wird mit den Funktionsbausteinen des TwinCAT Database Servers ein Funktionsbaustein erstellt, welcher verschiedene Methoden zum Erzeugen und Auslesen von Meldungen bereitstellt. Die Datenbank, in der die Meldungen abgelegt werden, wird aus der SPS heraus erzeugt. Im MAIN-Programm wird eine beispielhafte Verwendung des erzeugten Bausteins implementiert. Alle 7 Tage wird eine neue Datenbankdatei erzeugt. Drei verschiedene Meldungen können gesendet werden. Zusätzlich besteht die Möglichkeit die letzte Meldung bzw. alle Meldungen aus einem bestimmten Zeitraum abzurufen.



Kategorie	PLC Expert Mode
Verwendete Datenbank	MS Compact [▶ 131]
Kompatible Datenbanken	Mit kleineren Anpassungen für alle unterstützten Datenbanktypen anwendbar
Verwendete SPS Funktionsbausteine	FB_PLCDBCreateEvt [▶ 175], FB_PLCDBCmdEvt [▶ 187], FB_PLCDBWriteEvt [▶ 182], FB_PLCDBReadEvt [▶ 179]
Verwendete SPS Bibliotheken	Tc3_Database, Tc3_Eventlogger
Download	https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3495187979.zip

FB_ErrorLogger

CreateErrorLogDB (Methode)

Die Methode CreateErrorLogDB erzeugt eine MS-Compact-Datenbankdatei und erstellt die Tabelle, in die die Meldungen abgespeichert werden.

```
METHOD CreateErrorLogDB : BOOL
VAR_INPUT
    sDBName : T_MaxString;
END_VAR
```

```
CASE nState_CreateDB OF
    0:
        stDBConfig.sServer := CONCAT(CONCAT(sDBPath, sDBName), '.sdf');
        stDBConfig.bAuthentification := FALSE;

        nState_CreateDB := 1;
    1:
        IF fbPLCDBCreate.Database(pDatabaseConfig:= ADR(stDBConfig),
            cbDatabaseConfig:= SIZEOF(stDBConfig),
            bCreateXMLConfig:= FALSE, pDBID:= 0) THEN
            ipResultEvt := fbPLCDBCreate.ipTcResult;
            IF fbPLCDBCreate.bError THEN
                nState_CreateDB := 100;
```

```
            ELSE
                nState_CreateDB := 2;
            END_IF
        END_IF
    2:
        IF fbConfigTcDBSrv.Create(pTcDBSrvConfig:= ADR(stDBConfig),
            cbTcDBSrvConfig:= SIZEOF(stDBConfig), bTemporary:= TRUE,
             pConfigID:= ADR(nDBID) ) THEN
            ipResultEvt := fbConfigTcDBSrv.ipTcResult;
            IF fbConfigTcDBSrv.bError THEN
                nState_CreateDB := 100;
            ELSE
                nState_CreateDB := 3;
            END_IF
        END_IF
    3:
        arrTableColumns[0].sName := 'ID';
        arrTableColumns[0].eType := E_ColumnType.BigInt;
        arrTableColumns[0].nLength := 8;
        arrTableColumns[0].sProperty := 'IDENTITY(1,1)';

        arrTableColumns[1].sName := 'Timestamp';
        arrTableColumns[1].eType := E_ColumnType.DateTime;
        arrTableColumns[1].nLength := 4;

        arrTableColumns[2].sName := 'Severity';
        arrTableColumns[2].eType := E_ColumnType.NVarChar;
        arrTableColumns[2].nLength := 10;

        arrTableColumns[3].sName := 'ErrorCode';
        arrTableColumns[3].eType := E_ColumnType.Integer;
        arrTableColumns[3].nLength := 4;

        arrTableColumns[4].sName := 'Message';
        arrTableColumns[4].eType := E_ColumnType.NVarChar;
        arrTableColumns[4].nLength := 255;

        IF fbPLCDBCreate.Table(hDBID:= nDBID, sTableName:= sTableName,
            pTableCfg:= ADR(arrTableColumns),
            cbTableCfg:= SIZEOF(arrTableColumns)) THEN
            ipResultEvt := fbPLCDBCreate.ipTcResultEvent;
            nState_CreateDB := 100;
        END_IF
    100:
        IF _SetResultInfo(1033) THEN
            IF NOT bError THEN
                _bHasCreated := TRUE;
            END_IF

            nState_CreateDB := 0;
        END_IF
END_CASE

CreateErrorLogDB := nState_CreateDB = 0;
```

AddErrorEntry (Methode)

Mit der Methode AddErrorEntry können verschiedene Meldungen in die Datenbank geschrieben werden.

```
METHOD AddErrorEntry : BOOL
VAR_INPUT
    tTimestamp : DT;
    eSeverity : E_Severity;
    nErrCode : UDINT;
    sMessage : T_MaxString;
END_VAR
```

```
CASE nState_AddEntry OF
    0:
        ipResultEvt := fbPLCDBWrite.ipTcResult;

        stError.tTimestamp := tTimestamp;
        CASE eSeverity OF
            TcEventSeverity.Info:
                stError.sSeverity := 'Info';
            TcEventSeverity.Warning:
                stError.sSeverity := 'Warning';
            TcEventSeverity.Verbose:
                stError.sSeverity := 'Verbose';
            TcEventSeverity.Critical:
```

```
                stError.sSeverity := 'Critical';
            TcEventSeverity.Error:
                stError.sSeverity := 'Error';
        END_CASE
         stError.nErrCode := nErrCode;
         stError.sMsg := sMessage;

        arrColumns[0] := 'Timestamp';
        arrColumns[1] := 'ErrorCode';
        arrColumns[2] := 'Severity';
        arrColumns[3] := 'Message';

         nState_AddEntry := 1;
    1:
        IF fbPLCDBWrite.WriteStruct(
            hDBID:= nDBID,
            sTableName:= sTableName,
            pRecord:= ADR(stError),
            cbRecord:= SIZEOF(stError),
            pColumnNames:= ADR(arrColumns),
            cbColumnNames:= SIZEOF(arrColumns)) THEN

            nState_AddEntry := 100;
        END_IF
    100:
        IF _SetResultInfo(1033) THEN
            nState_AddEntry := 0;
        END_IF
END_CASE

AddErrorEntry := nState_AddEntry = 0;
```

ReadLastError (Methode)

Mit der Methode ReadLastError kann der aktuellste (letzte) Eintrag aus der Datenbank ausgelesen werden.

```
METHOD ReadLastError : BOOL
VAR_OUTPUT
    tTimestamp : DT;
    sSeverity : STRING(10);
    nErrCode : UDINT;
    sMessage : T_MaxString;
END_VAR
```

```
CASE nState_ReadLastEntry OF
    0:
        ipResultEvt := fbPLCDBRead.ipTcResult;

        arrColumns[0] := 'Timestamp';
        arrColumns[1] := 'ErrorCode';
        arrColumns[2] := 'Severity';
        arrColumns[3] := 'Message';

         nState_ReadLastEntry := 1;
    1:
        IF fbPLCDBRead.ReadStruct(
            hDBID:= nDBID,
            sTableName:= sTableName,
            pColumnNames:= ADR(arrColumns),
            cbColumnNames:= SIZEOF(arrColumns),
            sOrderByColumn:= 'ID',
            eOrderType:= E_OrderType.DESC,
            nStartIndex:= 0,
            nRecordCount:= 1,
            pData:= ADR(stReadData),
            cbData:= SIZEOF(stReadData)) THEN

            nState_ReadLastEntry := 100;
        END_IF
    100:
        IF _SetResultInfo(1033) THEN
            IF NOT fbPLCDBRead.bError THEN
                tTimestamp := stReadData.tTimestamp;
                sSeverity := stReadData.sSeverity;
                nErrCode := stReadData.nErrCode;
                sMessage := stReadData.sMsg;
            END_IF

            nState_ReadLastEntry := 0;
        END_IF
```

```
END_CASE

ReadLastError := nState_ReadLastEntry = 0;
```

GetErrorTimerange (Methode)

Wenn alle Meldungen eines bestimmten Zeitraums ausgelesen werden sollen, kann dies mit der Methode GetErrorTimerange erfolgen.

```
METHOD GetErrorTimerange : BOOL
VAR_INPUT
    tStartTimestamp : DT;
    tEndTimestamp : DT;
    nStartIndex : UDINT;
END_VAR
VAR_OUTPUT
    nErrorCount: UDINT;
    arrErrors : ARRAY [0..10] OF ST_ErrorEntry;
END_VAR
```

```
CASE nState_ErrorTimerange OF
    0:
        ipResultEvt := fbPLCDBRead.ipTcResult;

        stSearchData.dtStartTimestamp := tStartTimestamp;
        stSearchData.dtEndTimestamp := tEndTimestamp;

        sCmd := 'SELECT Timestamp, ErrorCode, Severity, Message FROM
         tbl_Errors WHERE Timestamp >= {start} AND Timestamp <= {end}';

        arrParameter[0].sParaName := 'start';
        arrParameter[0].eParaType := E_ExpParameterType.DateTime;
        arrParameter[0].nParaSize := 4;

        arrParameter[1].sParaName := 'end';
        arrParameter[1].eParaType := E_ExpParameterType.DateTime;
        arrParameter[1].nParaSize := 4;

        nState_ErrorTimerange := 1;
    1:
        IF fbPLCDBCmd.ExecuteDataReturn(
            hDBID:= nDBID,
            pExpression:= ADR(sCmd),
            cbExpression:= SIZEOF(sCmd),
            pData:= ADR(stSearchData),
            cbData:= SIZEOF(stSearchData),
            pParameter:= ADR(arrParameter),
            cbParameter:= SIZEOF(arrParameter),
            nStartIndex:= nStartIndex,
            nRecordCount:= 10,
            pReturnData:= ADR(arrErrs),
            cbReturnData:= SIZEOF(arrErrs),
            pRecords:= ADR(nErrCount)) THEN

            nState_ErrorTimerange := 100;
        END_IF
    100:
        IF _SetResultInfo(1033) THEN
            nErrorCount := nErrCount;
            arrErrors := arrErrs;

            nState_ErrorTimerange := 0;
        END_IF
END_CASE

GetErrorTimerange := nState_ErrorTimerange = 0;
```

_SetResultInfo (Private Methode)

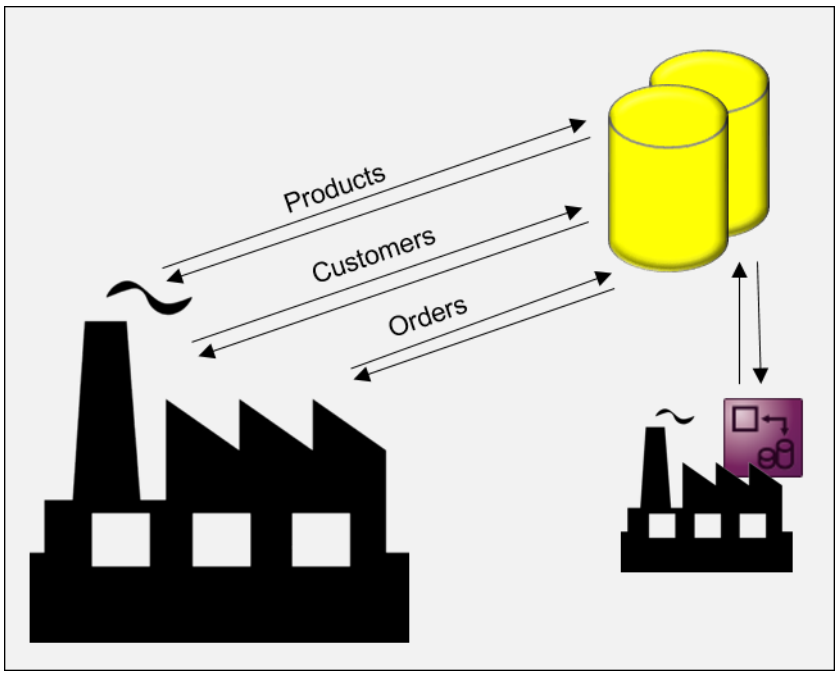
In der privaten Methode _SetResultInfo wird das Nachrichten Interface I_Message vom TwinCAT Eventlogger ausgewertet.

```
METHOD _SetResultInfo : BOOL
VAR_INPUT
    nLangId : INT := 1033;
END_VAR
```

```
_SetResultInfo := FALSE;

CASE nState_SetResInfo OF
    0:
        IF ipResultEvt.RequestEventText(nLangId, EventText, SIZEOF(EventText)) THEN
            nState_SetResInfo := 1;
        END_IF
    1:
        IF ipResultEvt.RequestEventClassName(nLangId, EventClassName, SIZEOF(EventClassName)) THEN
            EventSourcePath := ipResultEvt.ipSourceInfo.sName;
            EventId := ipResultEvt.nEventId;

            bError := (ipResultEvt.eSeverity = TcEventSeverity.Error) OR
                      (ipResultEvt.eSeverity = TcEventSeverity.Critical);

            nState_SetResInfo:=0;
            _SetResultInfo := TRUE;
        END_IF
END_CASE
```

7.1.1.3 Production Register

In diesem Szenario wird der SQL Expert Mode beispielhaft für die Handhabung mit gespeicherten Prozeduren (Stored Procedures) dargestellt. Aus der SPS heraus wird eine Verbindung zu einer Datenbank aufgebaut. Mithilfe von einer Stored Procedure werden aus mehreren Tabellen Produktpositionen ausgelesen. Die Bedienung erfolgt über eine Visualisierung.



Kategorie	SQL Expert Mode
Verwendete Datenbank	MS SQL [▶ 129]
Kompatible Datenbanken	MS SQL, MySQL, Oracle
Verwendete SPS Funktionsbausteine	FB_SQLDatabaseEvt [▶ 196], FB_SQLStoredProcedureEvt [▶ 205], FB_SQLResultEvt [▶ 203]
Verwendete SPS Bibliotheken	Tc3_Database, Tc3_Eventlogger
Download	https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3495190539.zip

In dem MAIN-Programm wird zur Abarbeitung eine sogenannte State-Machine implementiert, in der die verschiedenen SQL-Funktionsbausteine angesprochen werden. Da die Methoden der Funktionsbausteine kein Execute Flag besitzen, muss sichergestellt werden, dass die Methode nach Fertigstellung im nächsten Zyklus nicht noch einmal aufgerufen wird, da sonst der Vorgang wiederholt wird. Dies wird durch die State-Machine auf einfache Weise sichergestellt.

```
PROGRAM MAIN
VAR
    bCONNECT: BOOL;
    bEXECUTE: BOOL;
    bREAD : BOOL;
    bDISCONNECT: BOOL;

    R_TRIG1: R_TRIG;
    R_TRIG2: R_TRIG;
    R_TRIG3: R_TRIG;
    R_TRIG4: R_TRIG;

    nState: INT;
    nState_Connect: INT;
    nState_Disconnect: INT;

    bConn: BOOL;
    bSP: BOOL;
    bResult: BOOL;
    bData: BOOL;

    nDBID: UDINT := 1;

    fbSQLDatabase: FB_SQLDatabaseEvt(sNetID:='', tTimeout:=T#10S);
    fbSQLStoredProcedure: FB_SQLStoredProcedureEvt(
                        sNetID:='', tTimeout:=T#10S);
    fbSQLResult: FB_SQLResultEvt(sNetID:='', tTimeout:=T#10S);

    arrParameter: ARRAY [0..0] OF ST_SQLSPParameter;

    nCustomerID: DINT := 12345;

    nRecordStartIndex: UDINT;
    stRecordArr: ARRAY [1..20] OF ST_Record;
    nRecs: UDINT;

    ipResultEvt : Tc3_Eventlogger.I_TcMessage;
    bError : BOOL;
    nEventID: UDINT;
    sEventClass : STRING(255);
    sEventMsg : STRING(255);
END_VAR
```

```
R_TRIG1(CLK:=bCONNECT);
IF R_TRIG1.Q AND nState = 0 THEN
    nState := 1;
END_IF

R_TRIG2(CLK:=bEXECUTE);
IF R_TRIG2.Q AND nState = 0 THEN
    nState := 2;
END_IF

R_TRIG3(CLK:=bREAD);
IF R_TRIG3.Q AND nState = 0 THEN
    nState := 3;
END_IF

R_TRIG4(CLK:=bDISCONNECT);
IF R_TRIG4.Q THEN
    nState := 4;
END_IF
```

```
CASE nState OF
0:(*Idle*)
    ;
1: // Connect to database and create stored procedure instance
    CASE nState_Connect OF
        0:
            IF fbSQLDatabase.Connect(hDBID:= nDBID) THEN
                ipResultEvt := fbSQLDatabase.ipTcResult;
                bConn := NOT fbSQLDatabase.bError;
                IF bConn THEN
```

```
                    nState_Connect := 1;
                ELSE
                    nState:=200;
                END_IF
            END_IF
        1:
            arrParameter[0].sParameterName := '@Customer_ID';
            arrParameter[0].eParameterDataType :=
                            Tc3_Database.E_ColumnType.Integer;
            arrParameter[0].eParameterType := E_SPParameterType.Input;
            arrParameter[0].nParameterSize := SIZEOF(nCustomerID);

            IF fbSQLDatabase.CreateSP('SP_GetAddressByCustomerID',
                        ADR(arrParameter), SIZEOF(arrParameter),
                       ADR(fbSQLStoredProcedure)) THEN
                ipResultEvt:= fbSQLDatabase.ipTcResult;
                bSP := NOT fbSQLDatabase.bError;
                nState_Connect:=0;
                nState := 200;
            END_IF
    END_CASE
2: // Execute stored procedure
    IF fbSQLStoredProcedure.ExecuteDataReturn(
                        pParameterStrc:= ADR(nCustomerID),
                        cbParameterStrc:= SIZEOF(nCustomerID),
                        pSQLDBResult:= ADR(fbSQLResult)) THEN
        ipResultEvt:= fbSQLStoredProcedure.ipTcResult;
        MEMSET(ADR(stRecordArr),0,SIZEOF(stRecordArr));
        bResult := NOT fbSQLStoredProcedure.bError;
        nState := 200;
    END_IF
3:    // Read customer positions
    IF fbSQLResult.Read(nRecordStartIndex, 20, ADR(stRecordArr),
                    SIZEOF(stRecordArr), TRUE, FALSE) THEN
        ipResultEvt:= fbSQLResult.ipTcResult;
        bData := NOT fbSQLStoredProcedure.bError;
        nRecs := fbSQLResult.nDataCount;
        nState := 200;
    END_IF
4:// Disconnect all
    CASE nState_Disconnect OF
        0:
            IF bData THEN
                IF fbSQLResult.Release() THEN
                    nState_Disconnect := 1;
                END_IF
            ELSE
                nState_Disconnect := 1;
            END_IF
        1:
            IF bSP THEN
                IF fbSQLStoredProcedure.Release() THEN
                    nState_Disconnect := 2;
                END_IF
            ELSE
                nState_Disconnect := 2;
            END_IF
        2:
            IF bConn THEN
                IF fbSQLDatabase.Disconnect() THEN
                    nState_Disconnect := 3;
                END_IF
            ELSE
                nState_Disconnect := 3;
            END_IF
        3:
            bData := FALSE;
            bSP := FALSE;
            bConn := FALSE;
            bResult := FALSE;
            sEventClass := "";
            sEventMsg := "";
            nEventID := 0;
            bError := FALSE;
            nState_Disconnect := 0;
            nState := 0;
    END_CASE
200:
    IF ipResultEvt.RequestEventText(1033, sEventMsg, SIZEOF(sEventMsg)) THEN
        nState := 201;
```

```
    END_IF
201:
    IF ipResultEvt.RequestEventClassName(1033, sEventClass, SIZEOF(sEventClass)) THEN

        nEventID := ipResultEvt.nEventId;

        bError := (ipResultEvt.eSeverity = TcEventSeverity.Error) OR
                  (ipResultEvt.eSeverity = TcEventSeverity.Critical);

        nState:=0;
    END_IF
END_CASE
```

Die einzelnen Schritte des Ablaufs können in den einzelnen States in der SPS nachvollzogen werden. Zur einfachen Handhabung stehen boolsche Flags zur Verfügung.

1. bConnect: Verbindung wird mit der Datenbank aufgebaut
2. bExecute: Die Stored Procedure wird ausgeführt und Ergebnisse in den Zwischenspeicher geladen
3. bRead: Die Ergebnisse werden in die SPS übertragen
4. bDisconnect: Die Verbindung wird geschlossen

Werden diese Schritte nacheinander ausgeführt, wird das Array stRecordArr mit Werten aus der Datenbank gefüllt:

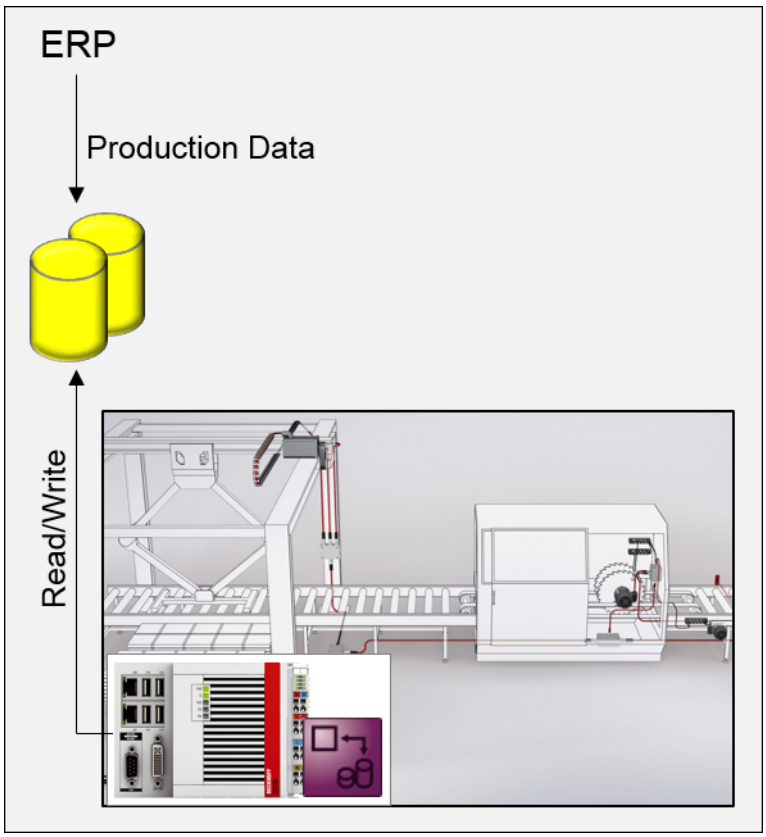
MAIN [Online] | MsSQL Customer DB | Sql Query Editor

TF6420_Sample3_CustomerProducts.CustomerProducts.MAIN

Expression	Type	Value	Prepared value
bCONNECT	BOOL	FALSE	TRUE
bEXECUTE	BOOL	FALSE	
bREAD	BOOL	FALSE	
bDISCONNECT	BOOL	FALSE	
stRecordArr	ARRAY [1..20] OF S...		
stRecordArr[1]	ST_Record		
nID	ULINT	12345	
sCustomer	STRING(50)	'Johann'	
sName	STRING(50)	'Groß'	
nProductNum	DINT	1	
sProductName	STRING(50)	'CX1000'	
sProductInfo	T_MaxString	'E'	
tTimestamp	DATE_AND_TIME	DT#2010-10-10-...	
stRecordArr[2]	ST_Record		
stRecordArr[3]	ST_Record		

7.1.1.4 Production Recipe

In diesem Szenario wird gezeigt, wie der TwinCAT Database Server mit beliebig strukturierten XML-Dateien umgeht. Aus einer XML-Datei wird das Produktionsrezept zum Bauen des Produkts ausgelesen und aus einer anderen Datei die entsprechenden Testparameter. Zudem werden die Testergebnisse in eine vorhandene XML-Datei geschrieben.



Kategorie	SQL Expert Mode
Verwendete Datenbank	XML [▶ 139] (als freie XML Dokumente)
Kompatible Datenbanken	XML
Verwendete SPS Funktionsbausteine	FB_PLCDBCmdEvt [▶ 187]
Verwendete SPS Bibliotheken	Tc3_Database, Tc3_Eventlogger
Download	https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3495193099.zip

Rezept XML:

```
<ProductionConfig>
    <Config TypeNum="12345" Test="985-524">
        <rLength>65.85</rLength>
        <rWidth>30</rWidth>
        <rHeight>2.5</rHeight>
        <iQuantity>500</iQuantity>
        <iCounter>0</iCounter>
    </Config>
    <Config TypeNum="23456" Test="985-524">
        <rLength>15.85</rLength>
        <rWidth>300</rWidth>
        <rHeight>12.5</rHeight>
        <iQuantity>1200</iQuantity>
        <iCounter>0</iCounter>
    </Config>
    <Config TypeNum="34567" Test="125-594">
        <rLength>195.85</rLength>
        <rWidth>378</rWidth>
        <rHeight>145.5</rHeight>
        <iQuantity>10</iQuantity>
        <iCounter>0</iCounter>
    </Config>
</ProductionConfig>
```

Test XML:

```
<ProductionConfig>
    <TestParameter>
        <Test Num="985-524">
            <MaxTemp>55.6</MaxTemp>
            <MinTemp>25.6</MinTemp>
            <MaxPSI>5500</MaxPSI>
        </Test>
        <Test Num="695-784">
            <MaxTemp>85.6</MaxTemp>
            <MinTemp>20.2</MinTemp>
            <MaxPSI>1300</MaxPSI>
        </Test>
        <Test Num="125-594">
            <MaxTemp>25.9</MaxTemp>
            <MinTemp>12.0</MinTemp>
            <MaxPSI>500</MaxPSI>
        </Test>
    </TestParameter>
    <Tests>
        <Test TestNum="985-524" TypeNum="12345" Timestamp="2016-09-24-06:00" Tester="Mustermann" Result="OK" />
    </Tests>
</ProductionConfig>
```

FB_ProductionConfigData

GetConfig (Methode)

Diese Methode liest das Produktionsrezept eines Produkts aus einer XML-Datei aus. Mithilfe von XPath Queries kann nach dem richtigen Rezept gesucht werden.

```
METHOD GetConfig : BOOL
VAR_INPUT
    nTypeNum : DINT;
END_VAR
VAR_OUTPUT
    stConfig : ST_Config;
END_VAR
```

```
GetConfig:= FALSE;

arrPara[0].sParaName := 'rLength';
arrPara[0].eParaType := Tc3_Database.E_ExpParameterType.Float32;
arrPara[0].nParaSize := 4;

arrPara[1].sParaName := 'rWidth';
arrPara[1].eParaType := Tc3_Database.E_ExpParameterType.Float32;
```

```
arrPara[1].nParaSize := 4;

arrPara[2].sParaName := 'rHeight';
arrPara[2].eParaType := Tc3_Database.E_ExpParameterType.Float32;
arrPara[2].nParaSize := 4;

arrPara[3].sParaName := 'iQuantity';
arrPara[3].eParaType := Tc3_Database.E_ExpParameterType.Int32;
arrPara[3].nParaSize := 4;

arrPara[4].sParaName := 'iCounter';
arrPara[4].eParaType := Tc3_Database.E_ExpParameterType.Int32;
arrPara[4].nParaSize := 4;

sCmd := CONCAT(CONCAT('XPATH_SEL<SUBTAG>#ProductionConfig/Config[@TypeNum
                    = ', DINT_TO_STRING(nTypeNum)), ']');

CASE nState_GetConfig OF
0:
    IF fbPLCDBCmd.ExecuteDataReturn(
        hDBID:= 1,
        pExpression:= ADR(sCmd),
        cbExpression:= SIZEOF(sCmd),
        pData:= 0,
        cbData:= 0,
        pParameter:= ADR(arrPara),
        cbParameter:= SIZEOF(arrPara[0])*5,
        nStartIndex:= 0,
        nRecordCount:= 1,
        pReturnData:= ADR(_stConfig),
        cbReturnData:= SIZEOF(_stConfig),
        pRecords:= 0) THEN

        ipResultEvt := fbPLCDBCmd.ipTcResult;
        nState_GetConfig := 100;
    END_IF
100:
    IF _SetResultInfo(1033) THEN
        GetConfig := TRUE;

        stConfig := _stConfig;

        nState_GetConfig := 0;
    END_IF
END_CASE
```

GetTestParameter (Methode)

Diese Methode liest die produktspezifischen Testparameter aus.

```
METHOD GetTestParameter : BOOL
VAR_INPUT
    nTypeNum : DINT;
END_VAR
VAR_OUTPUT
    sTestNum : STRING(8);
    stTestPara: ST_TestParameter;
END_VAR
```

```
GetTestParameter := FALSE;

CASE nState_GetTestPara OF
0:
    arrPara[0].sParaName := 'Test';
    arrPara[0].eParaType := Tc3_Database.E_ExpParameterType.STRING_;
    arrPara[0].nParaSize := 8;

    sCmd := CONCAT(CONCAT('XPATH_SEL<ATTR>#ProductionConfig/Config
                    [@TypeNum = ', DINT_TO_STRING(nTypeNum)), ']');

    IF fbPLCDBCmd.ExecuteDataReturn(
        hDBID:= 1,
        pExpression:= ADR(sCmd),
        cbExpression:= SIZEOF(sCmd),
        pData:= 0,
        cbData:= 0,
        pParameter:= ADR(arrPara),
        cbParameter:= SIZEOF(arrPara[0]),
        nStartIndex:= 0,
        nRecordCount:= 1,
```

```
        pReturnData:= ADR(_sTestNum),
        cbReturnData:= SIZEOF(_sTestNum),
        pRecords:= 0) THEN

        bError := fbPLCDBCmd.bError;
        sErrClass := fbPLCDBCmd.ipTcResultEvent.EventClassDisplayName;
        nErrID := fbPLCDBCmd.ipTcResultEvent.EventId;
        sErrText := fbPLCDBCmd.ipTcResultEvent.Text;

        IF fbPLCDBCmd .bError THEN
            ipResultEvt := fbPLCDBCmd.ipTcResult;
            nState_GetTestPara:= 100;
        ELSE
            nState_GetTestPara:= 1;
        END_IF
    END_IF
1:
    arrPara[0].sParaName := 'MaxTemp';
    arrPara[0].eParaType := Tc3_Database.E_ExpParameterType.Float32;
    arrPara[0].nParaSize := 4;

    arrPara[1].sParaName := 'MinTemp';
    arrPara[1].eParaType := Tc3_Database.E_ExpParameterType.Float32;
    arrPara[1].nParaSize := 4;

    arrPara[2].sParaName := 'MaxPSI';
    arrPara[2].eParaType := Tc3_Database.E_ExpParameterType.Int32;
    arrPara[2].nParaSize := 4;

    sCmd := CONCAT(CONCAT('XPATH_SEL<SUBTAG>#ProductionConfig/
                TestParameter/Test[@Num = $'', _sTestNum), '$']');

    IF fbPLCDBCmd.ExecuteDataReturn(
        hDBID:= 2,
        pExpression:= ADR(sCmd),
        cbExpression:= SIZEOF(sCmd),
        pData:= 0,
        cbData:= 0,
        pParameter:= ADR(arrPara),
        cbParameter:= SIZEOF(arrPara[0])*3,
        nStartIndex:= 0,
        nRecordCount:= 1,
        pReturnData:= ADR(_stTest),
        cbReturnData:= SIZEOF(_stTest),
        pRecords:= 0) THEN

        ipResultEvt := fbPLCDBCmd.ipTcResult;
        nState_GetTestPara:= 100;
100:
    IF _SetResultInfo(1033) THEN
        nState_GetTestPara := 0;

        stTestPara := _stTest;
        sTestNum := _sTestNum;

        GetTestParameter := TRUE;
    END_IF
END_CASE
```

AddTestEntry (Methode)

Diese Methode fügt das Testergebnis in die Test-XML-Datei ein.

```
METHOD AddTestEntry : BOOL
VAR_INPUT
    sTestNum : STRING(8);
    nTypeNum : DINT;
    sTimestamp : STRING;
    sTester : STRING;
    sResult : STRING;
END_VAR
```

```
AddTestEntry := FALSE;

arrPara[0].sParaName := 'TestNum';
arrPara[0].eParaType := Tc3_Database.E_ExpParameterType.STRING_;
arrPara[0].nParaSize := 8;

arrPara[1].sParaName := 'TypeNum';
arrPara[1].eParaType := Tc3_Database.E_ExpParameterType.Int32;
```

```
arrPara[1].nParaSize := 4;

arrPara[2].sParaName := 'Timestamp';
arrPara[2].eParaType := Tc3_Database.E_ExpParameterType.STRING_;
arrPara[2].nParaSize := 81;

arrPara[3].sParaName := 'Tester';
arrPara[3].eParaType := Tc3_Database.E_ExpParameterType.STRING_;
arrPara[3].nParaSize := 81;

arrPara[4].sParaName := 'Result';
arrPara[4].eParaType := Tc3_Database.E_ExpParameterType.STRING_;
arrPara[4].nParaSize := 81;

arrPara[5].sParaName := 'Test';
arrPara[5].eParaType := Tc3_Database.E_ExpParameterType.XMLTAGName;
arrPara[5].nParaSize := 0;

sCmd := 'XPATH_ADD<ATTR>#ProductionConfig/Tests';

stTest.sTestNum := sTestNum;
stTest.nTypeNum := nTypeNum;
stTest.sTimestamp := sTimestamp;
stTest.sTester := sTester;
stTest.sResult := sResult;

CASE nState_AddEntry OF
0:
    IF fbPLCDBCmd.Execute(
        hDBID:= 2,
        pExpression:= ADR(sCmd),
        cbExpression:= SIZEOF(sCmd),
        pData:= ADR(stTest),
        cbData:= SIZEOF(stTest),
        pParameter:= ADR(arrPara),
        cbParameter:= SIZEOF(arrPara)) THEN

        ipResultEvt := fbPLCDBCmd.ipTcResult;
        nState_AddEntry:= 100;
    END_IF
100:
    IF _SetResultInfo(1033) THEN
        nState_AddEntry:= 0;
        AddTestEntry:= TRUE;
    END_IF
END_CASE
```

_SetResultInfo (Private Methode)

In der privaten Methode _SetResultInfo wird das Nachrichten-Interface I_Message vom TwinCAT Eventlogger ausgewertet.

```
METHOD _SetResultInfo : BOOL
VAR_INPUT
    nLangId : INT := 1033;
END_VAR
```

```
_SetResultInfo := FALSE;

CASE nState_SetResInfo OF
    0:
        IF ipResultEvt.RequestEventText(nLangId, EventText, SIZEOF(EventText)) THEN
            nState_SetResInfo := 1;
        END_IF
    1:
        IF ipResultEvt.RequestEventClassName(nLangId, EventClassName, SIZEOF(EventClassName)) THEN
            EventId := ipResultEvt.nEventId;

            bError := (ipResultEvt.eSeverity = TcEventSeverity.Error) OR
                      (ipResultEvt.eSeverity = TcEventSeverity.Critical);

            nState_SetResInfo:=0;
            _SetResultInfo := TRUE;
        END_IF
END_CASE
```

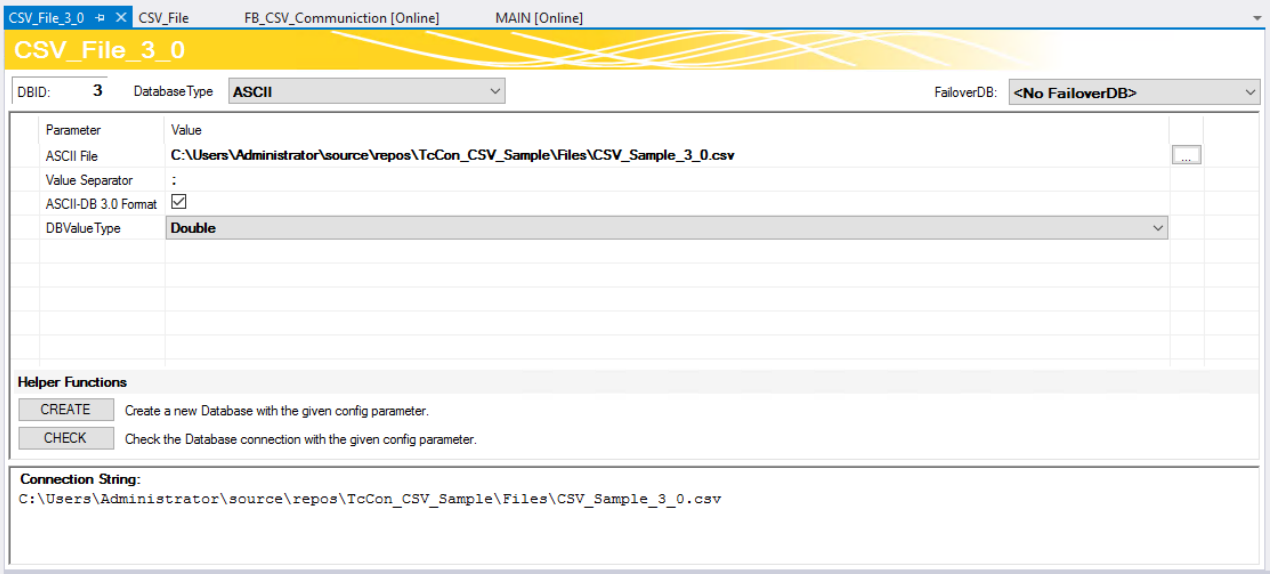
7.1.2 Best Practises

Die Tipps zur Verwendung des TwinCAT Database Server verdeutlichen die Vorteile der einzelnen Bausteine und Anwendungen hinsichtlich der Performance, Flexibilität und Komplexität.

7.1.2.1 CSV Dateien schreiben

Der TwinCAT 3 Database Server unterstützt das CSV-Dateiformat. Es gibt unterschiedliche Ansätze mit jeweils Vor- und Nachteilen, um Inhalte in die Datei zu schreiben oder zu lesen. Zwei dieser Ansätze werden hier genauer erläutert.

Wählen Sie die ASCII-Datenbank aus. Dort kann unter dem Dateipfad das Dateiformat .csv angegeben werden. Das ASCII-DB 3.0 Format-Flag gibt das Format der ASCII/CSV-Datei an. Fall das Format angehakt ist, wird das SAX-Verfahren genutzt. Mit dieser Einstellung ist der Schreibzugriff auf die Datei insbesondere mit dem FB_PLCDBCmdEvt-Baustein auch bei großen Dateien sehr performant. Ist das Format nicht angehakt, wird das DOM-Verfahren genutzt, welches sich besonders für das Lesen einer Datei eignet. Die Daten liegen dabei strukturiert im Arbeitsspeicher. Deshalb ist dieses Verfahren eher bei kleinere Dateien <1MB zu empfehlen. Durch die strukturierte Ablage bietet dieses Verfahren jedoch einige Vorteile. Die CSV-Datei kann mithilfe einer abgelegten Tabellenstruktur als SQL Datenbank genutzt werden. Nutzen Sie hierfür den SQL Query Editor. Über den ‚Create'-Button kann diese Datei direkt erzeugt werden.



Laden Sie ihre Konfiguration auf Ihr TwinCAT Database Server Target System.

Tab. 2: ASCII-Format Kompatibilität

Funktionsbaustein	Tabellen Struktur	ASCII-Format 3.0	Standard ASCII
FB_PLCDBWriteEvt.Write	standard	✓	✓
FB_PLCDBWriteEvt.WriteStruct*	beliebig	✗	✓
FB_PLCDBReadEvt.Read	standard	✓	✓
FB_:PLCDBReadEvt.ReadStruct*	beliebig	✗	✓
FB_PLCDBCmdEvt.Execute*	beliebig	✓	✗
FB_SQLCommandEvt	beliebig	✗	✓

*Markierte werden im folgenden Beispiel verwendet

Performant in die CSV-Datei schreiben

Die performanteste Art in eine CSV-Datei zu schreiben, ist die Nutzung des FB_PLCDBCmdEvt Bausteines. Dafür muss die Verknüpfung zu CSV-Datei im ASCII-DB 3.0 Format eingestellt werden. Der DBValueType spielt dabei keine Rolle. Eine Tabellenstruktur muss vorher nicht definiert werden.

Beispiel:

Gegeben sei folgende Struktur:

```
TYPE ST_CSVDataStruct :
STRUCT
    ID: LINT;
    Timestamp: DT;
    Name: STRING(80);
    Velocity: LREAL;
    Temperature: LREAL;
END_STRUCT
END_TYPE
```

Der Baustein wird folgendermaßen initialisiert:

```
VAR
    InputData: ST_CSVDataStruct;
    fbPLCDBCmd: FB_PLCDBCmd (sNetID:= '', tTimeout := T#30S);

    sCmd : T_MaxString := '{ID};{Timestamp};{Name};{Velocity};{Temperature}';

    para : ARRAY [0..4] OF ST_ExpParameter :=[
            (eParaType:= E_ExpParameterType.Int64, nParaSize := 8, sParaName := 'ID'),
            (eParaType:= E_ExpParameterType.DateTime, nParaSize := 4, sParaName := 'Timestamp'),
            (eParaType:= E_ExpParameterType.STRING_, nParaSize := 81, sParaName := 'Name'),
            (eParaType:= E_ExpParameterType.Double64, nParaSize := 8, sParaName := 'Velocity'),
            (eParaType:= E_ExpParameterType.Double64, nParaSize := 8, sParaName := 'Temperature')];
END_VAR
```

Die einzelnen Parameter werden in geschweiften Klammern im Kommando angegeben. Diese werden in über die Initialisierung beschrieben mit den Informationen zum Typ, der Bytelänge und dem Namen. Anhand des Namens wird der Parameter im Kommando wiedererkannt und entsprechend beim Schreiben in die Datei durch den Wert aus der PLC ersetzt.

Der Aufruf im PLC-Quellcode des Bausteins besteht aus einem Aufruf:

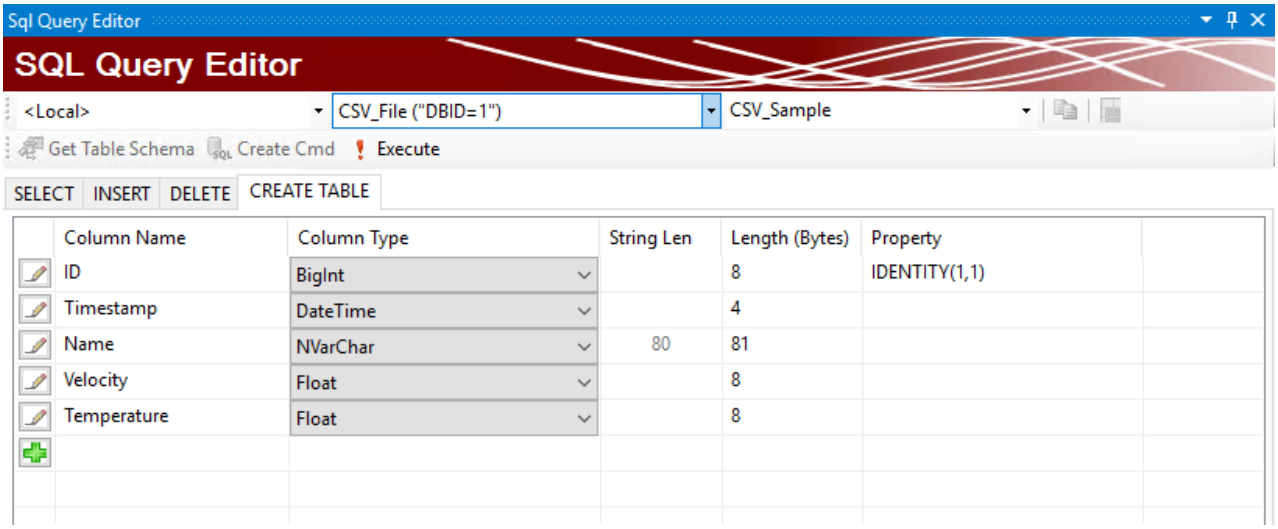
```
IF fbPLCDBCmd.Execute(
    hDBID:= 3,
    pExpression:= ADR(sCmd),
    cbExpression:= SIZEOF(sCmd),
    pData:= ADR(InputData),
    cbData:= SIZEOF(InputData) ,
    pParameter:= ADR(para),
    cbParameter:=SIZEOF(para))
THEN
    ;//Place for errorhandling or reactions;
END_IF
// Result: 16160;19-10-2018 12:27:38;Water Turbine;35.2238040741592;62.6461585412374
```

Die *hDBID* ist abhängig von ihrer Konfiguration und kann der Datenbankverknüpfung entnommen werden. Als *pData* (bzw. *cbData*) kann nicht nur die Adresse zur einzelnen Struktur, sondern auch zu einem Array ihrer Struktur angegeben werden. Dies kann nochmals zu Performanceverbesserungen führen.

Strukturiertes Schreiben und Lesen einer CSV-Datei

Nicht alle Funktionsbausteine sind mit dem ASCII-Format 3.0 möglich. Für einige Funktionen des TwinCAT Database Servers ist eine vorher konfigurierte Tabellenstruktur nötig. Diese lässt sich jedoch nicht im ASCII-Format 3.0 hinterlegen. In diesem Beispiel wird eine fest definierte Struktur verwendet, um die Daten mit den PLCDBWriteEvt- und PLCDBReadEvt-Bausteinen in beliebiger Struktur zu schreiben und zu lesen.

Folgende Struktur ist gegeben:



Auch als Export für die PLC unter dem ‚Select'-Tab möglich:

```
TYPE ST_CSVDataStruct :
STRUCT
    ID: LINT;
    Timestamp: DT;
    Name: STRING(80);
    Velocity: LREAL;
    Temperature: LREAL;
END_STRUCT
END_TYPE
```

Für beliebige Tabellenstrukturen werden die Write-/ReadStruct Methoden der jeweiligen PLC-Funktionsbausteine verwendet:

```
VAR
    fbPLCDBWrite: FB_PLCDBWrite(sNetID:= '', tTimeout := T#30S);
    fbPLCDBRead : FB_PLCDBRead(sNetID:= '', tTimeout := T#30S);
    ColumnNames : ARRAY [0..4] OF STRING(50) := ['ID','Timestamp','Name','Velocity','Temperature'];
    Data: ST_CSVDataStruct;
    ReadData: ARRAY[0..4] OF ST_CSVDataStruct;
END_VAR
```

```
IF fbPLCDBWrite.WriteStruct(
    hDBID:= hDBID,
    sTableName:= 'CSV_Sample',
    pRecord:= ADR(Data),
    cbRecord:= SIZEOF(Data),
    pColumnNames:= ADR(ColumnNames),
    cbColumnNames:= SIZEOF(ColumnNames) )
THEN
    ;//Place for errorhandling or reactions
END_IF

IF fbPLCDBRead.ReadStruct(
    hDBID:= hDBID,
    sTableName:= 'CSV_Sample',
    pColumnNames:= ADR(ColumnNames),
    cbColumnNames:= SIZEOF(ColumnNames) ,
    sOrderByColumn:= 'ID',
    eOrderType := E_OrderType.ASC,
    nStartIndex:= 0,
    nRecordCount:= 5,
    pData:= ADR(ReadData),
    cbData:=SIZEOF(ReadData))
THEN
    ;//Place for errorhandling or reactions
END_IF
```

Die Methode *WriteStruct(…)* schreibt die Struktur *Data* in die Datenbank. Anhand der *ColumnNames* werden die Strukturen der PLC und der CSV-Datei abgeglichen.

Die Methode *ReadStruct(…)* liest eine bestimmte Anzahl *(nRecordCount)* von Datensätzen aus der CSV-Datei. Diese können auch nach einer ausgewählten Spalte geordnet werden. Das Ziel-Array *ReadData* sollte dabei mindestens so groß sein, wie die Anzahl der abgeholten Daten, um alle Daten empfangen zu können.

Anhang

Eine Beispielkonfiguration beider Beispiele, sowie der vollständige Code eines einfachen Beispielprogramms können Sie hier herunterladen:https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/5778536715.zip. Um den Vorgang zu veranschaulichen, generiert das Programm Werte und schickt diese wiederholt in die CSV. Die oben verwendeten Einstellungen wurden dabei in einen eigenen Funktionsbaustein ausgelagert, welcher auf verschiedene Weise mit den beiden CSV-Formaten kommuniziert.

7.1.2.2 Schnelles Loggen mit Datenpuffer

Um Daten auch im Millisekunden Takt in eine Datenbank zu loggen, müssen diese zuvor zusammengefasst werden, bevor sie über den TwinCAT Database Server zur Datenbank übertragen werden. Diese sogenannten Datenpuffer können in ihrer Größe je nach Anforderung variieren. In dem Beispiel werden 100 Datensamples in einem Puffer zusammengefasst, bevor sie mit dem TwinCAT Database Server übertragen werden. Um keine Lücken durch den Schreibvorgang zu erhalten, müssen mehrere Puffer angelegt werden, in denen die Datensamples zusammengefasst werden. In dem Beispiel sind insgesamt 20 Puffer mit Hilfe eines 2-Dimensionalen Arrays angelegt.

Datensample

Definition:

```
TYPE ST_Data :
STRUCT
    Timestamp        : LINT;
    fAM              : LREAL;
    fPeak            : LREAL;
    fPulse           : LREAL;
    fSawtooth        : LREAL;
    fSine            : LREAL;
    fSquare          : LREAL;
    fStairs          : LREAL;
    fTriangular      : LREAL;
END_STRUCT
END_TYPE
```

Jeden Zyklus wird ein Element des Datenpuffers befüllt. Im Beispiel geschieht dies im 10ms Takt. Somit enthält ein Puffer Daten eines Zeitraums von 1s. Ist ein Puffer mit 100 Elementen gefüllt, wird durch ein weiteres Array signalisiert, dass die 100 Elemente nun mit dem Baustein FB_PLCDBCmdEvt übertragen werden können. Hierfür kann der gesamte Puffer dem Baustein übergeben werden. Jedes einzelne Element wird dann vom TwinCAT Database Server zur Datenbank übertragen. Dieses Beispiel kann auch mit anderen Bausteinen umgesetzt werden. Beachten Sie dabei, dass nicht alle Funktionsbausteine Arrays unterstützen.

Auszug aus dem Baustein FB_Record_tbl_Signals

(„State-Machine“ => State: Recording)

```
…
    2://Recording
        bRecording := TRUE;

        //Fill buffer
        stData[nWriteBufferIndex, nWriteIndex].Timestamp := nTimestamp;
        stData[nWriteBufferIndex, nWriteIndex].fAM := fAM;
        stData[nWriteBufferIndex, nWriteIndex].fPeak := fPeak;
        stData[nWriteBufferIndex, nWriteIndex].fPulse := fPulse;
        stData[nWriteBufferIndex, nWriteIndex].fSawtooth := fSawtooth;
        stData[nWriteBufferIndex, nWriteIndex].fSine := fSine;
        stData[nWriteBufferIndex, nWriteIndex].fSquare := fSquare;
        stData[nWriteBufferIndex, nWriteIndex].fStairs := fStairs;
        stData[nWriteBufferIndex, nWriteIndex].fTriangular := fTriangular;

        //Set buffer index
        nWriteIndex := nWriteIndex + 1;
        IF nWriteIndex = 100 THEN
            nWriteIndex := 0;
            aWriteSQL[nWriteBufferIndex]:= TRUE;
            nWriteBufferIndex := nWriteBufferIndex + 1;
```

```
            IF nWriteBufferIndex = 20 THEN
                nWriteBufferIndex := 0;
            END_IF

            IF aWriteSQL[nWriteBufferIndex] THEN
                nState := 255;
                RETURN;
            END_IF
        END_IF

        //Write buffer element (100 samples) to database
        IF aWriteSQL[nSQLIndex] THEN
            IF fbPLCDBCmd.Execute(nDBID, ADR(sCmd), SIZEOF(sCmd),
                                  ADR(stData[nSQLIndex,0]), SIZEOF(stData[nSQLIndex,0]) * 100,
                                  ADR(aPara), SIZEOF(aPara)) THEN
                IF fbPLCDBCmd.bError THEN
                    nState := 255;
                ELSE
                    nRecords := nRecords + 100;

                    aWriteSQL[nSQLIndex] := FALSE;

                    nSQLIndex := nSQLIndex + 1;
                    IF nSQLIndex = 20 THEN
                        nSQLIndex := 0;
                    END_IF

                    IF NOT bRecord THEN
                        bRecording := FALSE;
                        nState := 0;
                    END_IF
                END_IF
            END_IF
        END_IF
….
```

Anhang:

In diesem Best Practise wird mit Hilfe eines Funktionsgenerator Bausteins verschiedene Signale erzeugt, die in eine Datenbank geloggt werden können. Die Syntax des INSERT Kommandos ist allgemeingültig, wurde aber im speziellen mit einer MS SQL Datenbank getestet. Das unten angefügte ZIP enthält den kompletten Programmcode in Form eines Tnzip.

Download:https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/6263666699.zip

7.1.2.3 NoSQL

Dieses Dokument beschreibt den Umgang mit NoSql Datenbanken.

Genutzte Datenbank: MongoDB
Genutzter Datenbanktyp: DocumentDB

Schreiben von Daten

Datenbanktypen des Typs *DocumentDB* speichern Json-Dokumente beliebiger Struktur. Deshalb ist es möglich jegliche Struktur der SPS in *DocumentDB*s abzubilden. Dieses Dokument kann mithilfe des FB_JsonDataType automatisch erzeugt oder über die String-Bausteine zusammengebaut werden. Achten Sie darauf, dass die Dokumentvariable groß genug gewählt wird. Falls Sie mehrere Dokumente gleichzeitig hineinschreiben wollen, können Sie diese auch in einem Json-Array übergeben.

Zur Vorbereitung werden zunächst die QueryOptions [▶ 238] definiert. Dafür wird die betreffende Collection und der Abfragetyp angegeben. Für jeden Abfragetypen steht eine eigene Struktur zur Verfügung. Für das Schreiben von Dokumenten wird die Struktur T_QueryOptionDocumentDB_Insert [▶ 239] genutzt.

```
VAR
    fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
    InsertQueryOptions: T_QueryOptionDocumentDB_Insert;
    sDocument : STRING(2000);
END_VAR
```

```
InsertQueryOptions.pDocuments:= ADR(sDocument);
InsertQueryOptions.cbDocuments:= SIZEOF(sDocument);
fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.InsertOne;
fbNoSQLQueryBuilder_DocumentDB.sCollectionName := 'myCollection';
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions := ADR(InsertQueryOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(InsertQueryOptions);
```

Um das Dokument in die Datenbank zu schreiben wird der FB_NoSQLQueryEvt [▶ 212] verwendet. Mit der Execute() [▶ 213]-Methode werden die übergebenen Dokumente in die Datenbank geschrieben. Diese Ausführung verläuft asynchron zur SPS und kann mehrere Zyklen dauern. Der boolesche Rückgabewert signalisiert, wann der Baustein seinen Prozess abgeschlossen hat:

```
VAR
    fbNoSQLQuery: FB_NoSQLQueryEvt(sNetID := '', tTimeout := TIME#15S0MS);
    fbJsonDataType: FB_JsonReadWriteDatatype;
END_VAR
```

```
CASE eState OF
    …
    eMyDbState.Write:
```

```
        // set the document yourself as json format (Example)
        sDocument := '{"myBool" : true,
                       "Name" : "Some Name Value",
                       "Value": 2.3,
                       "Value2":3,
                       "Child":{"Name":"Single Child",
                                "Value":1,
                                "myBool":true,
                                "arr":[12.0,13.0,14.0,15.0],
                                "myBool2" : true},
                       "Children":[
                                {"Name":"Child1"
                                ,"Value": 1,
                                "myBool" : true,
                                "arr":[12.1,13.1,14.1,15.1],
                                "myBool2" : true},
                                {"Name":"Child2",
                                "Value":2,
                                "myBool" : true,
                                "arr":[12.2,13.2,14.2,15.2],
                                "myBool2" : true},
                                {"Name":"Child3",
                                "Value":1,
                                "myBool" : true,
                                "arr":[12.3,13.3,14.3,15.3],
                                "myBool2" : true}]
                       }';
```

```
        IF fbNoSQLQuery.Execute(1, myQueryBuilder) THEN
            IF fbNoSQLQuery.bError THEN
                InfoResult := fbNoSQLQuery.ipTcResult;
                eState:= eMyDbState.Error;
            ELSE
                eState:= eMyDbState.Idle;
            END_IF
        END_IF
    …
END_CASE
```

Die Datenbanken erkennen, mit welchem Datentyp die einzelnen Variablen gespeichert werden. Jedoch kann der Datentyp, wie bei *MongoDB,* explizit angegeben werden. Falls ein Timestamp explizit als Datentyp abspeichert werden soll, muss dieser im Json-Dokument definiert werden:

```
sDocument := '{…"myTimestamp": ISODate("2019-02-01T14:46:06.0000000"), …}';
```

Der String kann nicht nur über die String-Formatierungsbausteine der TwinCAT 3 Bibliotheken formatiert werden, sondern auch über Hilfsbausteine für Json-Dokumente, wie dem FB_JsonReadWriteDatatype aus der Tc3_JsonXml.

```
// set the document by JsonDataType
sTypeName := fbJsonDataType.GetDatatypeNameByAddress(SIZEOF(anyValue[1]), ADR(anyValue[1]));
sDocument := fbJsonDataType.GetJsonStringFromSymbol(sTypeName, SIZEOF(anyValue [1]), ADR(anyValue
[1]));
```

Lesen von Daten

Das Datenschema in der dokumentbasierten Datenbank kann für jedes Dokument unterschiedlich sein. Im Gegensatz dazu folgt die SPS ohne Weiteres einem festen Prozessabbild. Es kann sein, dass die Daten dem Prozessabbild nicht entsprechen.

In der Datenbank stehen zwei verschiedene Arten zur Verfügung, um Daten auszulesen. Die Find-Abfrage und die Aggregation. Beide liefern Ergebnisse aus der Datenbank zurück, wobei die Aggregation erweiterte Möglichkeiten bietet, die Daten in eine entsprechende Form zu bringen oder Operationen auszuführen, um beispielswese Mittelwerte direkt zu errechnen.

Zur Vorbereitung werden zunächst die QueryOptions [▶ 238] definiert. Dafür werden die betreffende Collection und der Abfragetyp angegeben. Für jeden Abfragetypen steht eine eigene Struktur zur Verfügung. Für das Aggregieren von Dokumenten wird die Struktur T_QueryOptionDocumentDB_Aggregation [▶ 238] genutzt.

```
VAR
    fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
    AggregationQueryOptions: T_QueryOptionDocumentDB_Aggregate;
    sPipeStages: STRING(1000);
END_VAR
```

```
AggregationQueryOptions.pPipeStages := ADR(sPipeStages);
AggregationQueryOptions.cbPipeStages := SIZEOF(sPipeStages);
fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.Aggregation;
fbNoSQLQueryBuilder_DocumentDB.sCollectionName := 'myCollection';
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions := ADR(AggregationQueryOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(AggregationQueryOptions);
```

Um die Aggregationsabfrage abzuschicken wird der FB_NoSQLQueryEvt [▶ 212] verwendet. Mit der ExecuteDataReturn() [▶ 214]-Methode werden die Parameter übermittelt und die zurückgelieferten Daten in die übergebene Speicherreferenz gelegt. Diese Ausführung verläuft asynchron zur SPS und dauert mehrere Zyklen. Der boolesche Rückgabewert signalisiert, wann der Baustein seinen Prozess abgeschlossen hat:

```
VAR
    fbNoSQLQuery: FB_NoSQLQueryEvt(sNetID := '', tTimeout := TIME#15S0MS);
    fbNoSQLResult: FB_NoSQLResultEvt(sNetID := '', tTimeout := TIME#15S0MS);
END_VAR
```

```
CASE eState OF
    …
    eMyDbState.Aggregation:
        sPipeStages :='{$$match :{}}';
        IF fbNoSQLQuery.ExecuteDataReturn(1, myQueryBuilder, pNoSqlResult:= ADR(fbNoSQLResult),
nDocumentLength=> nDocumentLength)) THEN
            IF fbNoSQLQuery.bError THEN
                InfoResult := fbNoSQLQuery.ipTcResult;
                eState:= eMyDbState.Error;
            ELSE
                eState:= eMyDbState.Idle;
            END_IF
        END_IF
    …
END_CASE
```

Die Syntax von *sPipeStages* hängt vom Datenbanktypen ab. Dieser wird alle Datensätze zurückliefern. Weitere exemplarische Möglichkeiten sind (mit fiktiven Datensätzen):

Operator	Beschreibung
{$$match : {Place : “NorthEast”}}	Alle Datensätze, welche „NorthEast“ als Wert des Elements „Place” tragen.
{$$project : { myValue : { $arrayElemAt : ["$WindPlantData.RotorSensor", 2]} } }	Liefert alle RotorSensor-Daten vom Arrayelementplatz 2 als „myValue“ zurück.
{$$project : {RotorAvg : {$avg: "$WindPlantData.RotorSensor"} } }	Liefert den Durchschnittswert des Datenarrays “RotorSensor” als „RotorAvg“ zurück.

Die vollständige Dokumentation der Operatoren finden Sie bei dem jeweiligen Datenbankhersteller.

Eine Referenz zu den zurückgelieferten Daten liegt nun im FB_NoSQLResultEvt [▶ 215]–Funktionsbaustein. Diese können nun als Json Dokumente in einen String oder als Struktur ausgelesen werden. Hier werden die Daten nun direkt in ein Array einer passenden Struktur gelesen. Über den SQL Query Editors des

Database Servers haben Sie die Möglichkeit direkt eine passende Struktur zu generieren, die zum Datensatz passt. Statt eines Arrays ist es auch möglich beim Abruf von nur einem Datensatz eine Adresse zu einer einzelnen Struktur zu hinterlegen.

```
VAR
    fbNoSQLResult: FB_NoSQLResultEvt(sNetID := '', tTimeout := TIME#15S0MS);
    aRdStruct : ARRAY [0..9] OF ST_MyCustomStruct;
    fbNoSqlValidation : FB_NoSQLValidationEvt(sNetID := '', tTimeout := TIME#15S0MS);
END_VAR
```

```
CASE eState OF
    …
    eMyDbState.ReadStruct:
        IF fbnoSQLDBResult.ReadAsStruct(0, 4, ADR(aRdStruct), SIZEOF(aRdStruct), bValidate := TRUE,
ADR(fbNoSqlValidation), bDataRelease:= TRUE) THEN
            IF fbnoSQLDBResult.bError THEN
                InfoResult := fbnoSQLDBResult.ipTcResult;
                eState:= eMyDbState.Error;
            ELSE
                eState:= eMyDbState.Idle;
            END_IF
        END_IF
    …
END_CASE
```

Der TwinCAT Database Server achtet bei der Zuordnung zwischen Datensatz und Struktur auf die Namen der Elemente im Datensatz und die Namen der Variablen. Sollen diese sich unterscheiden, kann in der SPS das Attribut „ElementName“ genutzt werden:

```
TYPE ST_WindFarmData :
STRUCT
    {attribute 'ElementName' := '_id'}
    ID: T_ObjectId_MongoDB;
    {attribute 'ElementName' := 'Timestamp'}
    LastTime: DT;
    {attribute 'ElementName' := 'WindPlantData'}
    Data: ST_WindFarmData_WindPlantData;
END_STRUCT
END_TYPE
```

„ElementName“ gibt in diesem Beispiel an, wie die Daten im Dokument der Datenbank heißen. Mit Hilfe des Startindizes und Angabe der Datensatzanzahl kann außerdem bei diesem Aufruf bestimmt werden, welche Datensätze zurückgeliefert werden sollen. Um mögliche Doppelungen zu vermeiden wird darauf hingewiesen, dass diese Optionen bereits mit Operatoren in den „PipeplineStages“ durchgeführt werden können.

Validieren von Daten

Falls es beim FB_NoSqlResult [▶ 215] Konflikte zwischen dem Datensatz und der Struktur in der SPS gab, können sie mit dem FB_NoSQLValidationEvt [▶ 219] ausgelesen werden. Konflikte können beispielsweise fehlende oder übrig gebliebene Datensätze, aber auch Datentypprobleme sein. Mit der Methode GetIssues() [▶ 220] können alle Konflikte als Array von Strings ausgelesen werden. Übrig gebliebene Daten, die nicht in der SPS-Struktur wiedergefunden wurden, können als Array von Strings im Json-Format über GetRemainingData() [▶ 221] ausgelesen werden. Gegebenenfalls können diese dann noch einmal separat in die richtige Struktur ausgelesen werden oder aber über die TwinCAT Json Bibliothek interpretiert werden.

```
VAR
    fbNoSqlValidation : FB_NoSQLValidationEvt(sNetID := '', tTimeout := TIME#15S0MS);
    aIssues : ARRAY[0..99] OF STRING(512);
    aRemaining : ARRAY [0..9] OF STRING(1000);
END_VAR
```

```
CASE eState OF
    …
    eMyDbState.ValidationIssues:
        IF fbValidation.GetIssues(ADR(aIssues), SIZEOF(aIssues), FALSE) THEN
            IF fbValidation.bError THEN
                InfoResult := fbValidation.ipTcResult;
                eState:= eMyDbState.Error;
            ELSE
                eState:= eMyDbState.Idle;
            END_IF

    eMyDbState.ValidationRemaining:
        IF fbValidation.GetRemainingData(ADR(aRemaining), SIZEOF(aRemaining), SIZEOF(aRemaining[1]),
bDataRelease:= FALSE)THEN
```

```
            IF fbValidation.bError THEN
                InfoResult := fbValidation.ipTcResult;
                eState:= eMyDbState.Error;
            ELSE
                eState:= eMyDbState.Idle;
            END_IF
    …
END_CASE
```

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/13743807627.zip

7.1.2.4 PostgreSql Routines

Neben den Functions (PostgreSQL Routine) werden seit PostgreSQL 11 auch Stored Procedures von der Datenbank unterstützt, um serverseitige Programmierung zu ermöglichen. Diese beiden Routine-Typen haben unterschiedliche Eigenschaften und Funktionen:

Tab. 3: Vergleich von Stored Procedures und Functions

	Stored Procedures	Functions
OUT Parameter	+	-
Rückgabewert	-	+
Kann in Abfragen genutzt werden	-	+
Unterstützt Transaktionen	+	-

Diese Eigenschaften erfordern unterschiedliche Schnittstellen mit dem TwinCAT Database Server. Wie bei anderen unterstützten Datenbanken, werden Stored Procedures über den Funktionsbaustein FB_SQLStoredProcedureEvt [▶ 205] ausgeführt. Functions können in SQL-Befehle eingebaut werden, welche über den FB_PLCDBCmdEvt [▶ 187] oder den FB_SQLCommandEvt [▶ 200] aufgerufen werden.

PostgreSQL nutzt für die Rückgabe von Datensätzen der Routinen „RefCursor“. Der TwinCAT Database Server wertet diese „RefCursor“ automatisch aus und gibt den darin referenzierten Datensatz zurück. Die Auflösung multipler „RefCursor“ ist nicht möglich.

Aufruf einer Stored Procedure

Procedures können über den FB_SQLStoredProcedureEvt aufgerufen werden.

Beispiel (SQL)

SQL Script zum Erstellen einer Procedure:

```
CREATE OR REPLACE PROCEDURE "public"."SP_getLastData"(
    INOUT result_data refcursor)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    open result_data for SELECT * FROM "myTable_Double" LIMIT 10;
END
$BODY$;
```

Falls die Prozedur einen (oder mehrere) „RefCursor“ als Ausgangsparameter definiert, wird dieser (bzw. der erste) automatisch interpretiert und die resultierenden Datensätze für den FB_SQLResultEvt [▶ 203] in den Zwischenspeicher abgelegt. Der Datentyp „RefCursor“ wird vom TwinCAT Database Server wie ein String behandelt.

Beispiel (TwinCAT 3 in ST)

```
VAR
    fbSqlDatabase   : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
    ParaInfo        : ST_SQLSPParameter;
END_VAR
```

```
ParaInfo.sParameterName     := '@result_data';
ParaInfo.eParameterType     := E_SPParameterType.InputOutput;
ParaInfo.eParameterDataType := E_ColumnType.RefCursor; // 19
ParaInfo.nParameterSize     := 81;
```

```
IF fbSQLDatabase.CreateSP('"public"."SP_getLastData"', ADR(ParaInfo), SIZEOF(ParaInfo), ADR(fbSQLSto
redProcedure)) THEN
    IF fbSQLDatabase.bError THEN
        nState:=255;
    ELSE
        nState:= nState+1;
    END_IF
END_IF
```

Der FB_SQLStoredProcedureEvt [▶ 205] nutzt die zuvor mit FB_SQLDatabaseEvt.CreateSP() [▶ 196] verknüpfte Stored Procedure

```
VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID:='', tTimeout := T#5S);
    sRefCursor            : STRING;
    tcMessage            : I_TcMessage;
END_VAR
```

```
IF fbSQLStoredProcedure.ExecuteDataReturn(pParameterStrc := ADR(sRefCursor),
cbParameterStrc:= SIZEOF(sRefCursor), pSQLDBResult := ADR(fbSqlResult)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLResultEvt [▶ 203] genutzt werden, um die Daten auszulesen.

Aufruf einer Function

Functions können innerhalb von SQL Befehlen aufgerufen werden.

Beispiel (SQL)

SQL Script zum Erstellen einer Function:

```
CREATE OR REPLACE FUNCTION "public"."F_getLastData"()
    RETURNS refcursor
    LANGUAGE 'plpgsql'
AS $BODY$
DECLARE result_data refcursor;
BEGIN
    open result_data for SELECT * FROM "myTable_Double" ORDER BY "ID" DESC LIMIT 10;
    return result_data;
END;$BODY$;
```

Zum Aufrufen der Function wird folgender SQL Befehl verwendet:

```
SELECT "public"."F_getLastData"();
```

Der Aufruf selbst gibt einen „RefCursor“ zurück. Dieser wird vom TwinCAT Database Server automatisch interpretiert.

Beispiel (TwinCAT 3 in ST)

Der FB_SQLCommandEvt [▶ 200] nutzt das vom FB_SQLDatabaseEvt.CreateCmd() [▶ 196] erstellte Kommando.

```
VAR
    fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
    tcMessage    : I_TcMessage;
END_VAR
```

```
sCmd := 'SELECT "public"."getLastData"();';

// call sql command
IF fbSQLCommand.ExecuteDataReturn(ADR(sCmd), SIZEOF(sCmd), ADR(fbSqlResult)) THEN
    IF fbSQLCommand.bError THEN
        tcMessage := fbSQLCommand.ipTcResult;
        nState := 255; // error state
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Nachfolgend kann der FB_SQLResultEvt [▶ 203] genutzt werden, um die Datensätze auszulesen.

Es wird empfohlen diesen Programmcode in einer State Machine zu nutzen.

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/13743810955.zip

7.1.2.5 Zyklische Daten und Zeitreihen-Datenbanken

Dieses Dokument beschreibt den Umgang mit Zeitreihen und wie zyklisch auftretende Daten in Zeitreihendatenbanken gespeichert werden.

Genutzte Datenbank: InfluxDB
Genutzter Datenbanktyp: TimeSeriesDB

Einleitung

Das Schreiben von Daten in regelmäßigen oder zyklischen Abständen ist ein häufiger Anwendungsfall in der Steuerungstechnik. Dabei sollten die Daten zeitgenau erfasst werden. Da Datenbankkommunikation nicht echtzeitfähig ist, ist es sinnvoll die regelmäßig gemessenen Daten in einem Zwischenspeicher zu speichern. Dafür kann ein Array von der Datenstruktur dienen. Die gesammelten Daten werden daraufhin an den TwinCAT Database Server gesendet und können dort zeitunkritisch verarbeitet und schließlich in der Datenbank gespeichert werden.

Zeit

In der verwendeten Datenbank werden Datensätze jeweils mit einem Zeitstempel abgespeichert. Zusammen mit den Tag-Spalten bilden diese eine eindeutige Identifikation. Falls zwei Datensätze eine identische Identifikation (gleiche Zeitstempel- und Tag-Werte) tragen, überschreibt der neuere den alten Datensatz.

Beispiel:

	time	locationname (tag)	temperature (field)	windspeed(field)
1	1581675200630326200	Verl	11.5	6.3
2	1581675200630327200	Verl	10.3	5.2
3	1581675200630328200	Verl	9.8	2.8
4	1581675200630328200	Hamburg	14.2	14.9
4	1581675200630328200	Hamburg	15.6	8.9
…	…	…	…	…

Datensatz Nr. 4 wird von neuem Datensatz überschrieben, da die Identifikation identisch ist.

Der Zeitstempel in der Datenbank wird standardmäßig als UNIX-Epoch-Zeit abgespeichert. Mit Ausnahme, der selbst erstellten Insert-Befehle werden die Zeitstempel bei den Funktionsbausteinen des TwinCAT 3 Database Servers als TwinCAT-Zeit (Anzahl der 100ns-Schritten seit 1.1.1601) entgegengenommen und konvertiert. Im Falle von Insert-Befehlen werden die Zeiten nicht konvertiert.

Datenbankkonfiguration

In der Datenbankkonfiguration sollte InfluxDB ausgewählt sein. Fügen Sie die Verbindungsparameter zur gewünschten Datenbank ein. Falls noch keine Datenbank vorhanden sein sollte, können Sie die Datenbank über den „Create“-Button erzeugen. Achten Sie auf Ihre Firewall-Einstellungen bzw. Portfreigaben. **Eine Tabelle muss nicht angelegt werden, da diese automatisch beim ersten Zugriff von InfluxDB erzeugt wird.** Ein festes Tabellenschema gibt es bei InfluxDB nicht. Auch nachträglich können Spalten erweitert oder hinzugefügt werden.

Schreiben zyklischer Daten

Dieses Beispiel soll zeigen, wie mit minimiertem Aufwand Symbole aus der SPS in eine Zeitseriendatenbank geschrieben werden können.

Deklaration:

```
    State: E_DbLogState;
    bWriting: BOOL; // Set this bool fla to write the data once into the InfluxDB

    dbid: UDINT := 1; // Handle to the configured database

    QueryOption_TSDB_Insert : T_QueryOptionTimeSeriesDB_Insert; // defines detailed Queryparameter
    fbNoSQLQueryBuilder_TimeSeriesDB : FB_NoSQLQueryBuilder_TimeSeriesDB; // defines database type
specific api
    fbNoSqlQueryEvt : FB_NoSQLQueryEvt(sNetID := '', tTimeout := T#15S); // functionblock to execute
queries

    // databuffer for 1 second with 10 ms time delta
    windTurbineData: ARRAY[1..100] OF WindTurbineData;

    // error handling helper values
    TcResult: Tc3_Database.I_TcMessage;
    bError: BOOL;
    sErrorMessage: STRING(255);

    i: INT;
    rand : DRAND;
    nrand: LREAL;
```

Deklarieren der Datenquellenstruktur:

In dieser Struktur werden die Attribute „TagName“ und „FieldName“ verwendet, um die Datenfelder als Tags oder Fields zu deklarieren. Standardmäßig werden sie als Fields deklariert. Falls der Spaltenname in der Tabelle vom Symbolnamen in der SPS abweichen soll, können ebenfalls diese Attribute verwendet werden.

Um nicht gesetzte Daten bei der Datenanalyse zu erfassen, können Standardwerte außerhalb des Wertebereichs verwendet werden, um diese bei der Analyse zu erkennen.

```
TYPE WindTurbineData :
STRUCT

    {attribute 'TagName' := 'ID'}
    WindTurbineID : STRING(255);

    {attribute 'FieldName' := 'Power'}
    Power : LREAL := -1; // [0) [kW]

    {attribute 'FieldName' := 'Wind Speed'}
    WindSpeed : LREAL := -1; // [0) [m/s]

    {attribute 'FieldName' := 'Wind Direction'}
    WindDirection : LREAL := -1; // [0,360][°]

END_STRUCT
END_TYPE
```

(WindTurbineData.tcDUT)

Deklaration des ENUM für die State-Machine:

```
TYPE E_DbLogState :
(
    idle := 0,
    init,
    writing,
    error
);
END_TYPE
```

Generieren von Beispieldaten:

```
        FOR i := 1 TO 100 BY 1 DO
             rand();
             nrand := rand.Num;
            windTurbineData[i].WindDirection := 240.328 + nrand;
            windTurbineData[i].WindSpeed := 7.3292 + nrand;
            windTurbineData[i].Power := 1133.1975 + nrand;
            windTurbineData[i].WindTurbineID := 'Wind Turbine Verl 13';
        END_FOR

CASE State OF

    E_DbLogState.idle:
        IF bWriting THEN
            bWriting := FALSE;
            State := E_DbLogState.init;
        END_IF
```

Vorbereiten des Aufrufs:

In diesem Fall wird das Array ‘windTurbineData’ in die Tabelle ‚WindMeasurement‘ der Datenbank geschrieben. Dafür werden die Daten direkt aus dem Prozessabbild gelesen. Um die Adressen im Speicher auszulesen wird der Datentyp angegeben. Die Zeit der Datensätze wird automatisch unter Angabe der Startzeit und dem Zeitabstand generiert. Die Daten müssen dafür korrekt im Array abgespeichert werden. Pro SPS-Zyklus kann beispielsweise ein Datensatz im Array genutzt werden. Es ist sinnvoll für diesen Prozess eine SPS-Task anzulegen. In diesem Bespiel beträgt die Zykluszeit 10 ms.

```
    E_DbLogState.init:

        fbNoSQLQueryBuilder_TimeSeriesDB.pQueryOptions := ADR(QueryOption_TSDB_Insert);
        fbNoSQLQueryBuilder_TimeSeriesDB.cbQueryOptions := SIZEOF(QueryOption_TSDB_Insert);

        QueryOption_TSDB_Insert.sTableName := 'WindMeasurement';
        QueryOption_TSDB_Insert.sDataType := 'WindTurbineData';
        QueryOption_TSDB_Insert.pSymbol := ADR(windTurbineData);
        QueryOption_TSDB_Insert.cbSymbol := SIZEOF(windTurbineData);
        QueryOption_TSDB_Insert.nDataCount := 100;
        QueryOption_TSDB_Insert.nStartTimestamp := F_GetSystemTime();
        QueryOption_TSDB_Insert.nCycleTime := 10000; // (in 100 ns)
        State := E_DbLogState.writing;
```

Schreiben der Daten:

Dieser Aufruf schreibt die schreibt die Daten in die konfigurierte Datenbank mit der entsprechenden Datenbank-ID. Dieser kann mehrere Zyklen dauern, da es sich um einen asynchronen Prozess handelt. Gegebenenfalls müssen mehrere Speicherarrays verwendet werden, um eine lückenlose Aufzeichnung der Daten zu gewährleisten.

```
    E_DbLogState.writing:

        IF fbNoSqlQueryEvt.Execute(dbid, fbNoSQLQueryBuilder_TimeSeriesDB) THEN
            IF fbNoSqlQueryEvt.bError THEN
                TcResult := fbNoSqlQueryEvt.ipTcResult;
                State := E_DbLogState.error;
            ELSE
                State := E_DbLogState.idle;
            END_IF
        END_IF
```

Fehlerbehandlung:

Nutzen Sie den Tc3-Eventlogger für Ihre Fehlerbehandlung

```
    E_DbLogState.error:

        IF TcResult.RequestEventText(1033, sErrorMessage, SIZEOF(sErrorMessage)) THEN
            TcResult.Send(F_GetSystemTime());
            State := E_DbLogState.idle;
            bError := TRUE;
        END_IF


END_CASE
```

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/13743809291.zip

7.2 Tc2_Database

Alle Beispielanwendungen für den TwinCAT Database Server sind in einer Solution zusammengefasst. Die Solution kann hier an zentraler Stelle heruntergeladen werden: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip

Die zip-Datei beinhaltet neben der tszip-Datei für die TwinCAT 3 Solution alle notwendigen dateibasierten Datenbanken. Wenn der Ordner „Samples“ aus der zip-Datei in den Standard-Installationsordner: *C:\TwinCAT\Functions\TF6420-Database-Server\Win32* gepackt wird, müssen die Pfade in der Database-Server-Konfiguration nicht weiter editiert werden. Die Beispiele mit nicht dateibasierten Datenbanken, wie eine MS SQL, müssen mit dem Konfigurator jedoch individuell angepasst werden.

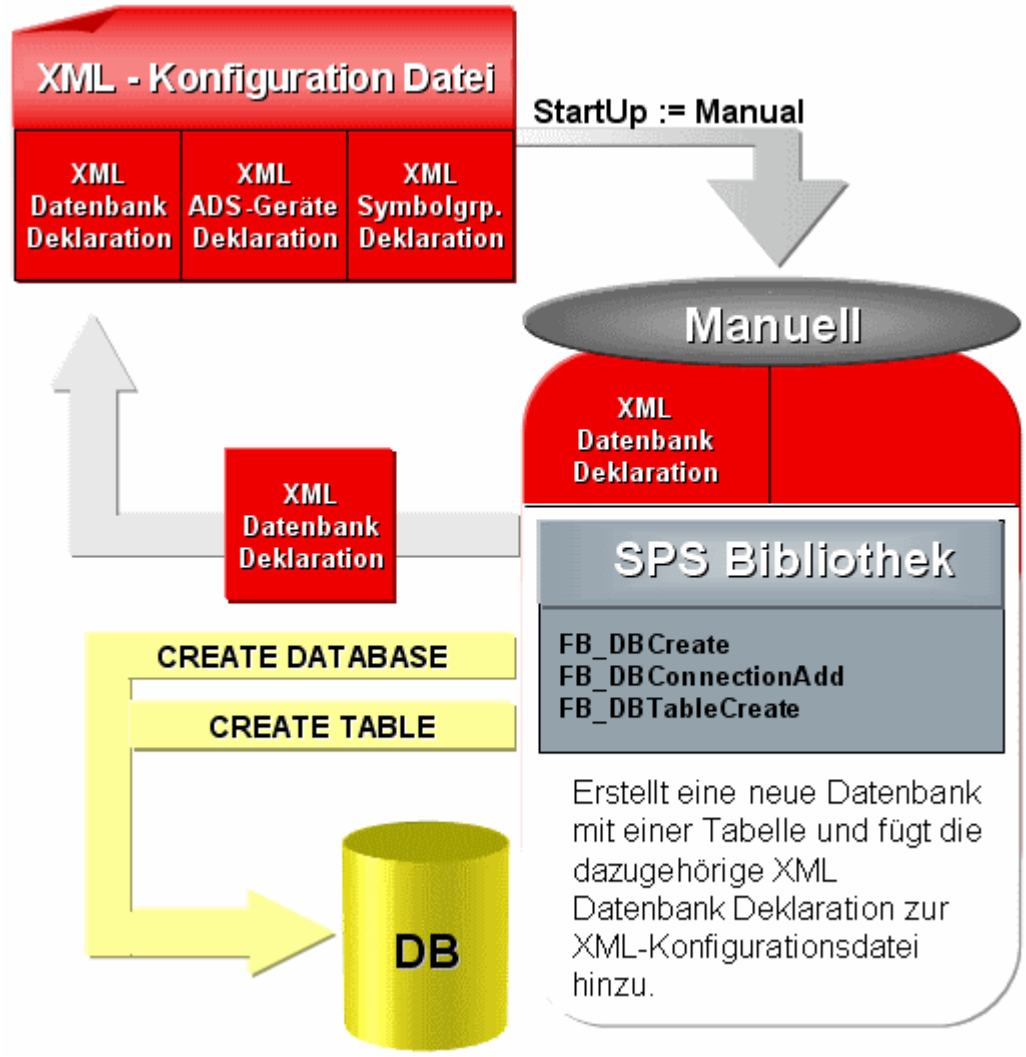
Die einzelnen Beispiele sind auf eigenen Seiten ausführlich dokumentiert:

SP-Projektname	Beschreibung
Create_DB_Sample [▶ 364]	Erstellen einer Datenbankverbindung und Tabelle aus der SPS heraus
Cyclic_RdWrt_Sample [▶ 367]	Zyklisches Loggen/Schreiben in/aus einer Datenbank
Write_DB_Sample [▶ 369]	Schreiben von Variablen in eine Datenbank mit einem einfach SPS-Baustein ohne SQL-Kommando
SQL_InsertSelect_Sample [▶ 373]	Beispiel mit FB_DBRecordInsert/FB_DBRecordArraySelect Baustein
StoredProcedures_Sample [▶ 375]	Gespeicherte Prozeduren (Stored Procedures) mit FB_DBStoredProceduresRecordArray
XML_DB_Sample [▶ 378]	XML-Dateien als Datenbank nutzen
XML_XPath_Sample [▶ 383]	XML-XPath-Beispiel ohne Schema
XML_XPath_Schema_Sample [▶ 386]	XML-XPath-Beispiel mit XML-Schema - Vergleichbar mit TwinCAT XML Server "Read"

7.2.1 Erstellen einer MS Access Datenbank

In diesem Beispiel wird gezeigt, wie eine Datenbank aus der SPS heraus erstellt wird. Zusätzlich wird eine Tabelle hinzugefügt und die erzeugte Datenbank in der XML-Konfigurationsdatei deklariert.

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip



Verwendeter Datenbanktyp	MS Access
Kompatible Datenbanktypen	MS SQL, MS Compact SQL, MS Access, XML
Verwendete Funktionsbausteine	FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate
Einzubindende Bibliotheken	Tc2_Database, Tc2_System, Tc2_Standard
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip

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_BIGINT),
     (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_BIGINT),
     (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*)
        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.

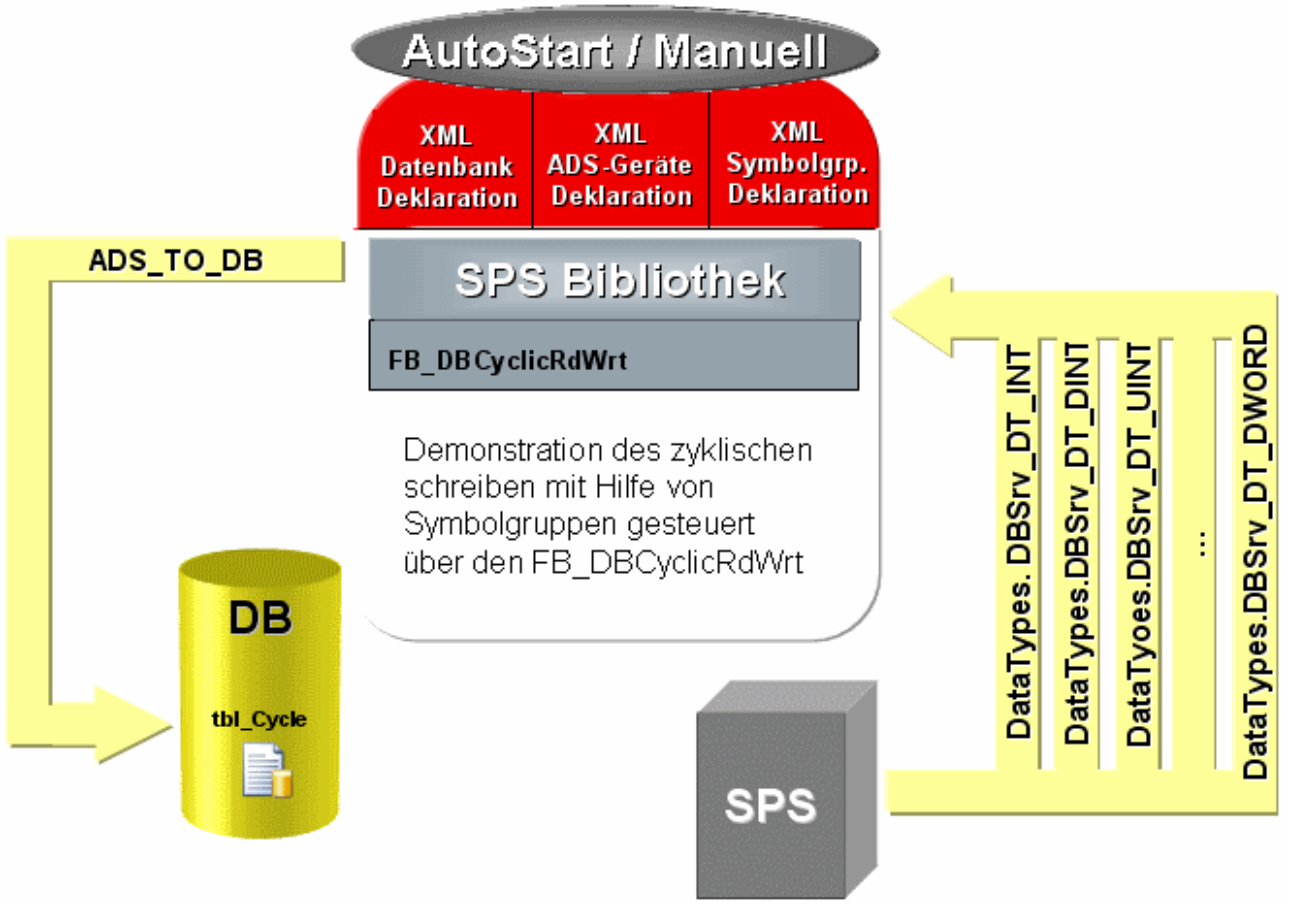
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

7.2.2 Starten / Stoppen zyklisches Loggen

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

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip



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	Tc2_Database, Tc2_System, Tc2_Standard
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip, 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_STRUCT
END_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.

TwinCAT Database Server

Alle Microsoft SQL Compact Datenbanken, die in der 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.

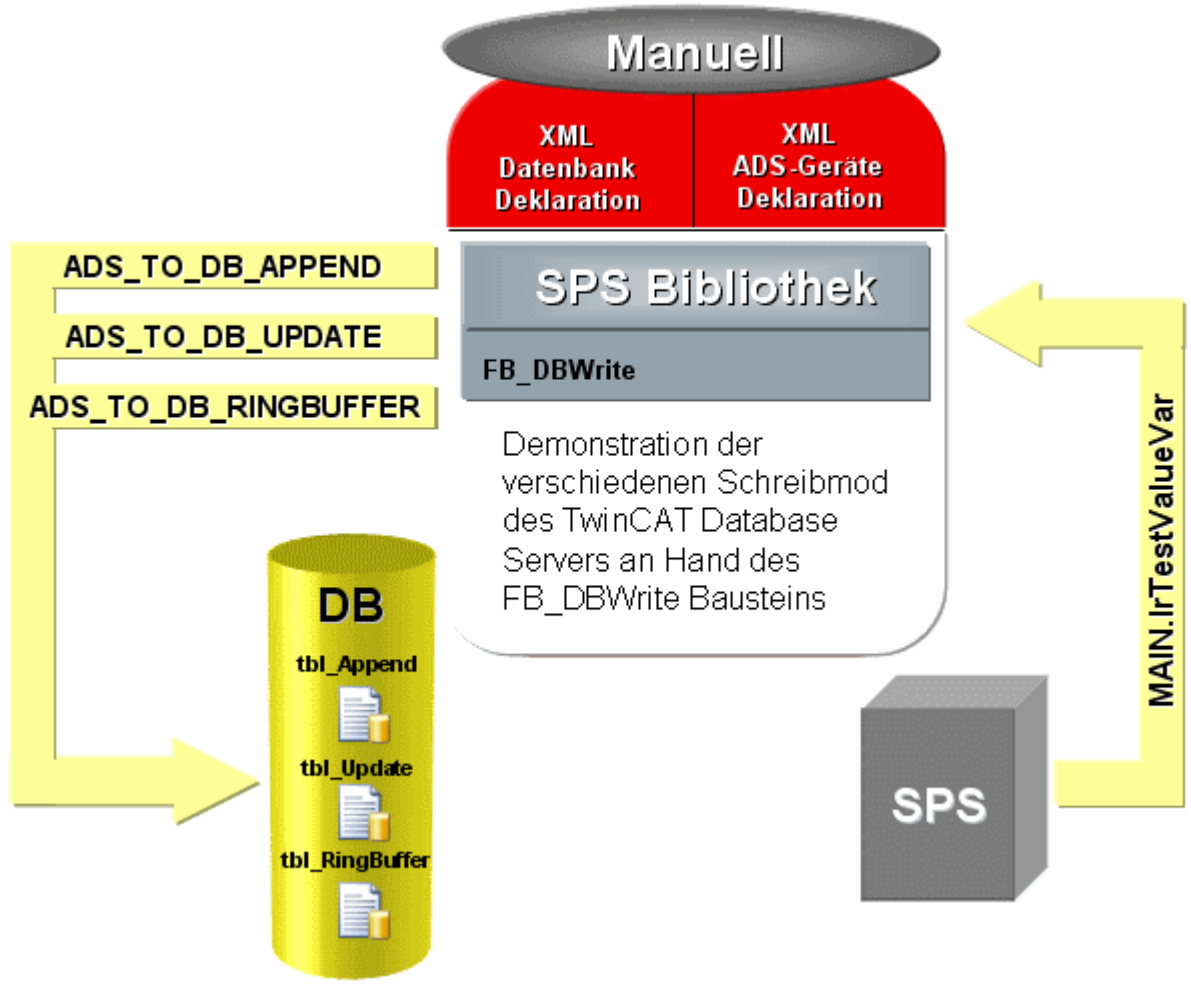
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

7.2.3 Loggen einer SPS-Variablen mit FB_DBWrite

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

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip



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	Tc2_Database, Tc2_System, Tc2_Standard
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml, SQLQuery.sql

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

Beispiel Aufbau:

Mithilfe der Variable „eWriteMode“ kann eingestellt werden, mit welchem 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 will 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 records 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_IF
END_CASE
```

TwinCAT Database Server

Alle Microsoft SQL Compact Datenbanken, die in der 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.

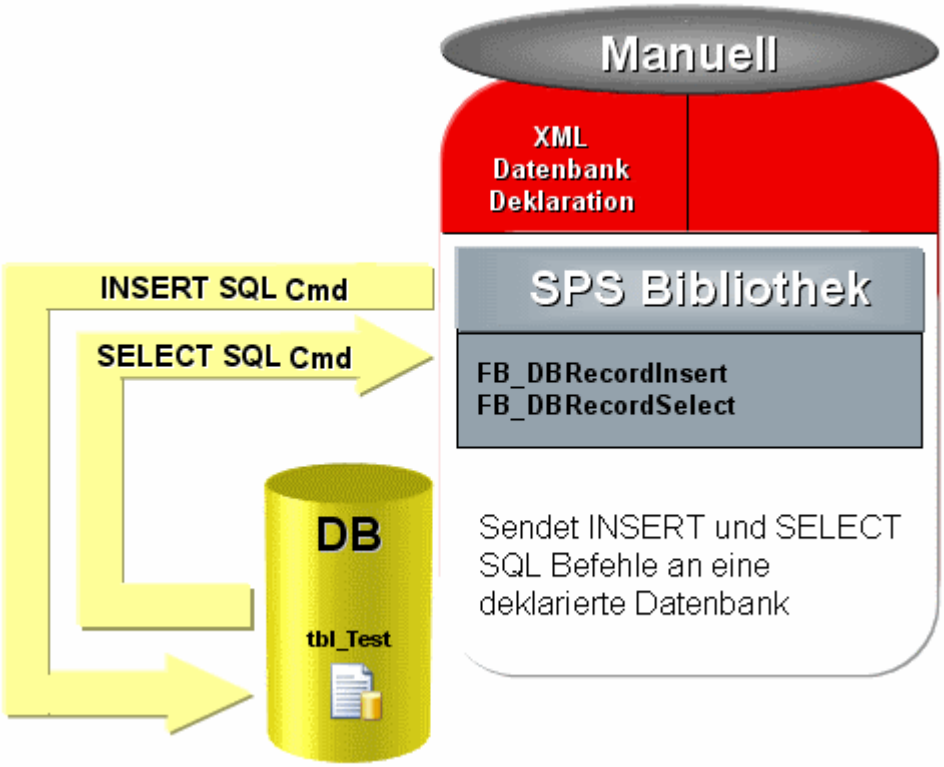
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

7.2.4 Beispiel mit dem FB_DBRecordInsert und FB_DBRecordSelect Baustein

In diesem Beispiel gezeigt, wie mehrere Werte in eine Datenbank aus der SPS heraus mit dem Funktionsbaustein FB_DBRecordInsert geloggt werden. 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.

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip



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	"**Tc2_Database**", "Tc2_System", "Tc2_Standard", "Tc2_Utilities"
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml, TestDB_Access.mdb

In folgende Tabellenstruktur wird geschrieben:

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

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

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

Struct ST_Record

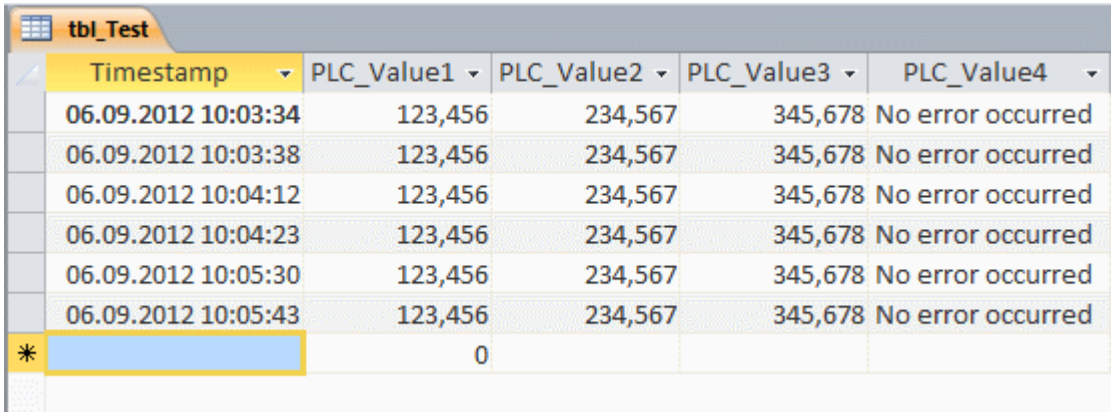
```
TYPEST_Record :
STRUCT
    Timestamp : DT;
    PLC_Value1: REAL;
    PLC_Value2: REAL;
    PLC_Value3: REAL;
    PLC_Value4: STRING;
END_STRUCT
END_TYPE
```

SPS-Programm

```
CASEeState 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
```

Um dieses Beispiel zu benutzen, müssen Sie die Access Datenbank „Sample7.mdb“ in der XML-Konfigurationsdatei deklarieren.
Durch die Erzeugung einer positiven Flanke an der Variable „bStartstopInsert“ wird ein Datensatz mit den vier SPS-Werten und dem Timestamp in der Datenbank angelegt.



tbl_Test

Timestamp	PLC_Value1	PLC_Value2	PLC_Value3	PLC_Value4
06.09.2012 10:03:34	123,456	234,567	345,678	No error occurred
06.09.2012 10:03:38	123,456	234,567	345,678	No error occurred
06.09.2012 10:04:12	123,456	234,567	345,678	No error occurred
06.09.2012 10:04:23	123,456	234,567	345,678	No error occurred
06.09.2012 10:05:30	123,456	234,567	345,678	No error occurred
06.09.2012 10:05:43	123,456	234,567	345,678	No error occurred
*	0			

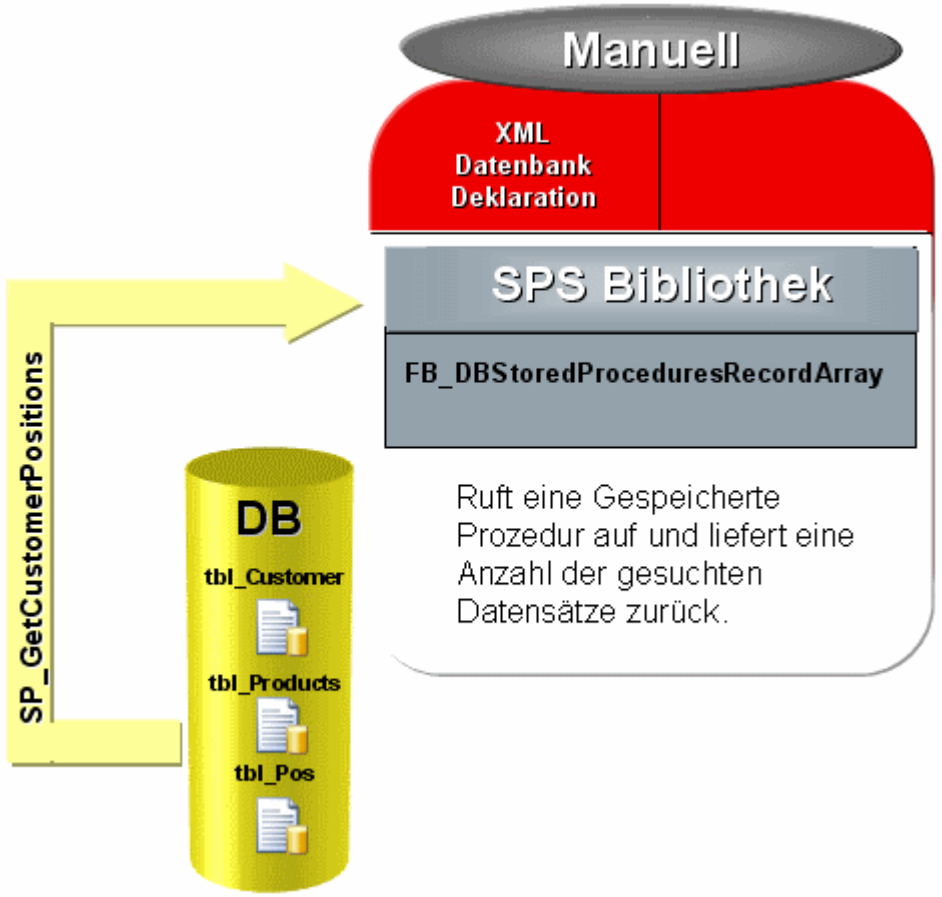
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

7.2.5 Stored Procedures mit FB_DBStoredProceduresRecordArray

Mithilfe des Funktionsbausteins FB_DBStoredProceduresRecordArray können Parameter als INPUT, OUTPUT oder INOUT deklariert werden und den gespeicherten Prozeduren (Stored Procedures) übergeben werden. So können komplexe SQL-Kommandos am Datenbank-Server vorprogrammiert und dann vom TwinCAT Database Server getriggert werden. Im Gegensatz zum FB_DBStoredProceduresRecordReturn-Baustein können mehrere Datensätze mit einem Aufruf zurückgeliefert werden.

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip



Verwendeter Datenbanktyp	MS SQL (MS SQL Server 2008)
Kompatible Datenbanktypen	MS SQL, MySQL, Oracle
Verwendete Funktionsbausteine	FB_DBStoredProceduresRecordArray
Einzubindende Bibliotheken	Tc2_Database, Tc2_System, Tc2_Base, Tc2_Utilities
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml

Das folgende Beispiel zeigt den Aufruf einer einfachen gespeicherten Prozedur (Stored Procedure) mit einem Eingangsparameter und Rückgabedatensatz. Die Prozedur wurde an einem Microsoft SQL Server 2008 erstellt.

Code der Stored Procedure 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;

    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
```

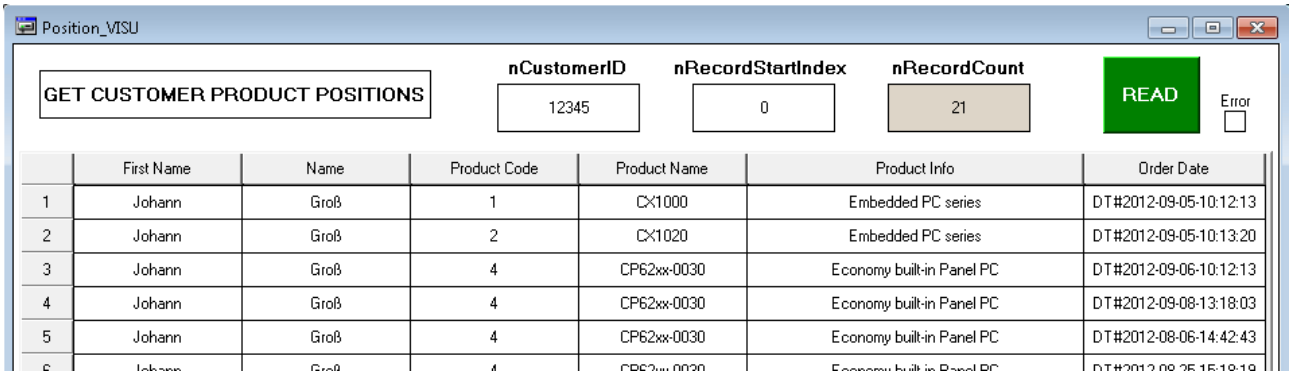
Datensatzstruktur in der SPS (ST_Record)

```
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_IF

CASE 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_IF
END_CASE
```

Visualisierung



	First Name	Name	Product Code	Product Name	Product Info	Order Date
1	Johann	Groß	1	CX1000	Embedded PC series	DT#2012-09-05-10:12:13
2	Johann	Groß	2	CX1020	Embedded PC series	DT#2012-09-05-10:13:20
3	Johann	Groß	4	CP62xx-0030	Economy built-in Panel PC	DT#2012-09-06-10:12:13
4	Johann	Groß	4	CP62xx-0030	Economy built-in Panel PC	DT#2012-09-08-13:18:03
5	Johann	Groß	4	CP62xx-0030	Economy built-in Panel PC	DT#2012-08-06-14:42:43

Voraussetzungen

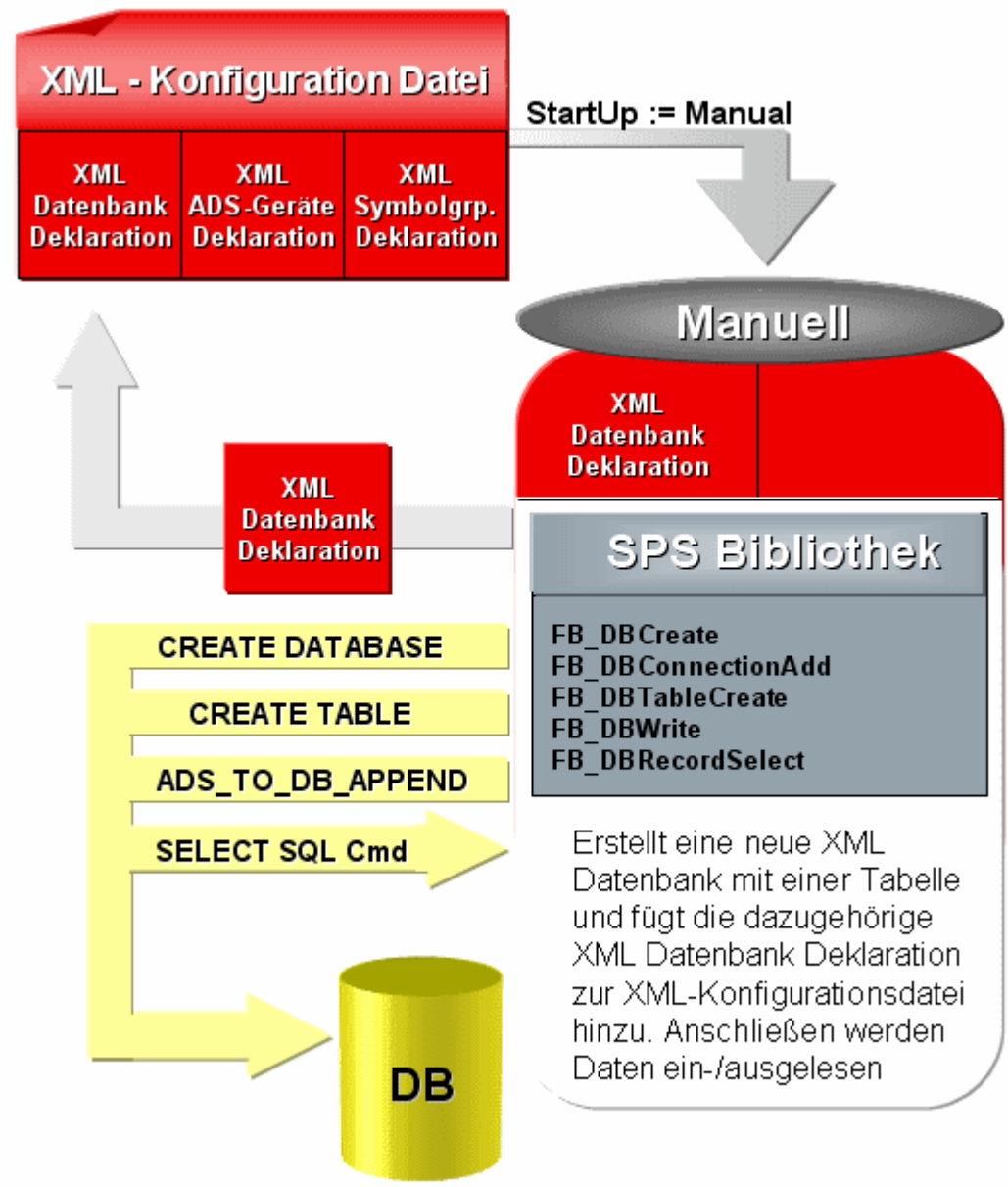
Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

7.2.6 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 gezeigt, 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: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.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	"**Tc2_Database**", "Tc2_System", "Tc2_Standard", "Tc2_Utilities"
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip

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 FB_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 FB_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 NOTbBusy_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;
            IFnCounter = 5 THEN
                nState     := 6;
```

```
            END_IF
        END_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_IF
    END_CASE
```

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

Folgende Dateien werden erzeugt:

XMLTestDB.xml (XML Datenbank Datei)

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

XMLTestDB.xsd (XML Schema)

```
<?xmlversion="1.0"?>
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleTypename="bigint">
    <xsd:restrictionbase="xsd:long" />
  </xsd:simpleType>
  <xsd:simpleTypename="datetime">
    <xsd:restrictionbase="xsd:dateTime" />
  </xsd:simpleType>
  <xsd:simpleTypename="ntext_80">
    <xsd:restrictionbase="xsd:string">
      <xsd:maxLengthvalue="80" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleTypename="float">
    <xsd:restrictionbase="xsd:double" />
  </xsd:simpleType>
  <xsd:complexTypename="myTable_Type">
    <xsd:sequence>
      <xsd:elementminOccurs="0"maxOccurs="unbounded"name="row">
        <xsd:complexType>
          <xsd:attributename="ID"type="bigint" />
          <xsd:attributename="Timestamp"type="datetime" />
          <xsd:attributename="Name"type="ntext_80" />
          <xsd:attributename="Value" type="float" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:elementname="XMLTestDB">
    <xsd:complexType>
      <xsd:sequenceminOccurs="1"maxOccurs="1">
        <xsd:elementname="myTable"type="myTable_Type" />
      </xsd:sequence>
```

```
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

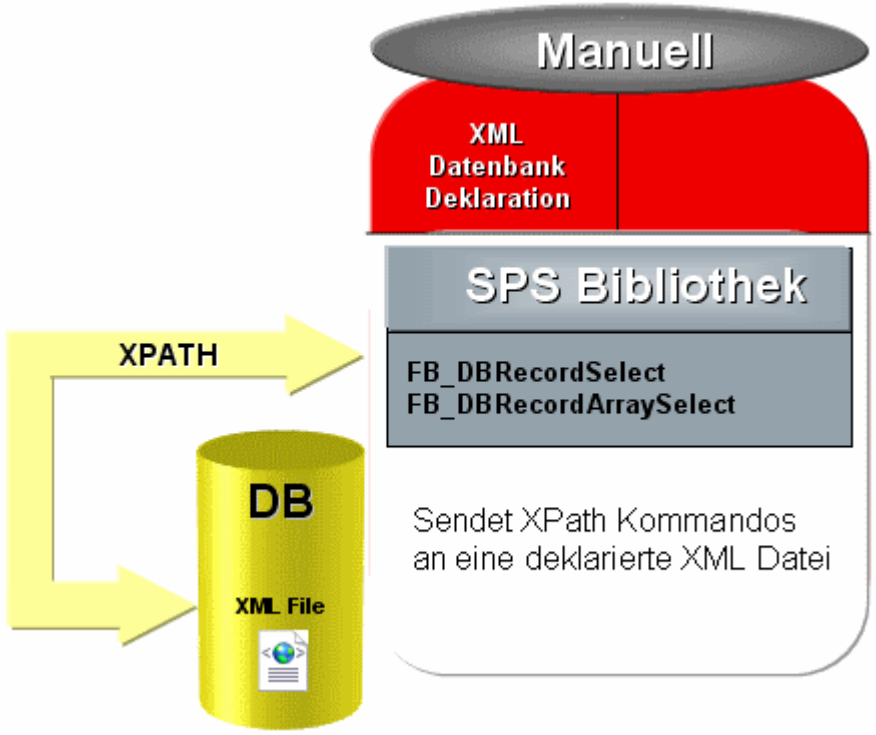
Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

7.2.7 XPath-Beispiel zur Darstellung der unterschiedlichen SELECT-Typen

Mithilfe des Funktionsbausteins FB_DBRecordArraySelect/FB_DBRecordSelect können XPath-Kommandos abgesetzt und XML-Tags aus einer beliebigen XML-Datei gelesen werden. In diesem Beispiel wird gezeigt, wie mithilfe des TwinCAT Database Servers unterschiedliche Einträge aus XML-Dateien gelesen werden können. Es werden einzelne Tags, SubTags und auch das Auslesen von Attributen unterstützt und dargestellt.

Download: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip



Verwendeter Datenbanktyp	XML
Kompatible Datenbanktypen	XML
Verwendete Funktionsbausteine	FB_DBRecordArraySelect
Einzubindende Bibliotheken	Tc2_Database, Tc2_System, Tc2_Standard, Tc2_Utilities
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip

Beispiel XML-Datei (XMLFactoryXY.xml)

```
<?xmlversion="1.0" encoding="utf-8" ?>
<Factory_XY>
<Name>Sample Factory XY</Name>
<Factory_Info>
<Street>Samplestreet 25</Street>
<City>33415 Verl</City>
<Country>Germany</Country>
<Office_Count>1</Office_Count>
```

```
<Employe_Count>6</Employe_Count>
<Manager>Max Mustermann</Manager>
</Factory_Info>
<Employees>
<Employeeid="10001" name="Julia Kingston" department="Development" position="Worker"
hired="2001-08-01" />
<Employeeid="10002" name="Jens Marx" department="Import" position="Worker" hired="2003-08-01" />
<Employeeid="10003" name="Justus Kaiser" department="Export" position="Worker" hired="2003-08-01" />
<Employeeid="10004" name="Marc Klein" department="Production" position="Worker" hired="2005-08-01" /
>
<Employeeid="10005" name="Matt Bloomberg" department="Production" position="Worker"
hired="2005-08-01" />
<Employeeid="10006" name="Frida Hundt" department="Production" position="Worker"
hired="2010-08-01" />
</Employees>
</Factory_XY>
```

Struktur ST_FactoryInfo

```
TYPEST_FactoryInfo :
STRUCT
 sStreet       : T_MaxString;
 sCity         : T_MaxString;
 sCountry      : T_MaxString;
 sOffice_Count : T_MaxString;
 sEmploye_Count: T_MaxString;
 sManager      : T_MaxString;
END_STRUCT
END_TYPE
```

Struktur ST_Employee

```
TYPEST_Employee :
STRUCT
 sID          : T_MaxString;
 sName        : T_MaxString;
 sDepartment  : T_MaxString;
 sPosition    : T_MaxString;
 sHired       : T_MaxString;
END_STRUCT
END_TYPE
```

MAIN-Programm

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

     nState    : INT;

     sXPath    : T_MaxString;

     fbDBRecordArraySelect : FB_DBRecordArraySelect;

     bBusy_ReadFactoryName : BOOL;
     bError_ReadFactoryName: BOOL;
     nErrID_ReadFactoryName: UDINT;

     bBusy_ReadFactoryInfo : BOOL;
     bError_ReadFactoryInfo: BOOL;
     nErrID_ReadFactoryInfo: UDINT;

     bBusy_ReadEmployee    : BOOL;
     bError_ReadEmployee   : BOOL;
     nErrID_ReadEmployee   : UDINT;

     stSQLState            : ST_DBSQLError;

     sFactoryName          : T_MaxString;
     stFactoryInfo         : ST_FactoryInfo;
     aEmployees            : ARRAY [1..10] OF ST_Employee;
END_VAR
```

```
R_TRIG1(CLK:=bSTART);
IF R_TRIG1.Q THEN
     bSTART:=FALSE;
     fbDBRecordArraySelect(bExecute:=FALSE);
     nState:=1;
END_IF
```

```
CASE nState OF
     0://IDLE
         ;
     1://Read Factory Name
         sXPath:= 'XPATH#Factory_XY/Name';
         fbDBRecordArraySelect(
             sNetID           := ,
             hDBID            := 7,
             pCmdAddr         := ADR(sXPath),
             cbCmdSize        := SIZEOF(sXPath),
             nStartIndex      := 0,
             nRecordCount     := 1,
             pDestAddr        := ADR(sFactoryName),
             cbRecordArraySize:= SIZEOF(sFactoryName),
             bExecute         := TRUE,
             tTimeout         := T#15S,
             bBusy             => bBusy_ReadFactoryName,
             bError            => bError_ReadFactoryName,
             nErrID            => nErrID_ReadFactoryName,
             sSQLState         => stSQLState,
             nRecords          => );

         IF NOT bBusy_ReadFactoryName THEN
             fbDBRecordArraySelect(bExecute:=FALSE);
             IF NOT bError_ReadFactoryName THEN
                 nState        :=2;
             ELSE
                 nState        :=255;
             END_IFEND_IF
     2://Read Factory Info
         sXPath                 := 'XPATH#Factory_XY/Factory_Info';
         fbDBRecordArraySelect(
             sNetID           := ,
             hDBID            := 7,
             pCmdAddr         := ADR(sXPath),
             cbCmdSize        := SIZEOF(sXPath),
             nStartIndex      := 0,
             nRecordCount     := 1,
             pDestAddr        := ADR(stFactoryInfo),
             cbRecordArraySize := SIZEOF(stFactoryInfo),
             bExecute         := TRUE,
             tTimeout         := T#15S,
             bBusy             => bBusy_ReadFactoryInfo,
             bError            => bError_ReadFactoryInfo,
             nErrID            => nErrID_ReadFactoryInfo,
             sSQLState         => stSQLState,
             nRecords          => );

         IF NOT bBusy_ReadFactoryInfo THEN
             fbDBRecordArraySelect(bExecute:=FALSE);
             IF NOT bError_ReadFactoryInfo THEN
                 nState        :=3;
             ELSE
                 nState        :=255;
             END_IF
         END_IF
     3://Read Employees
         sXPath                  := 'XPATH#Factory_XY/Employees/Employee';
         fbDBRecordArraySelect(
             sNetID           := ,
             hDBID            := 7,
             pCmdAddr         := ADR(sXPath),
             cbCmdSize        := SIZEOF(sXPath),
             nStartIndex      := 0,
             nRecordCount     := 10,
             pDestAddr        := ADR(aEmployees),
             cbRecordArraySize  := SIZEOF(aEmployees),
             bExecute         := TRUE,
             tTimeout         := T#15S,
             bBusy             => bBusy_ReadEmployee,
             bError            => bError_ReadEmployee,
             nErrID            => nErrID_ReadEmployee,
             sSQLState         => stSQLState,
             nRecords          => );

         IF NOT bBusy_ReadEmployee THEN
             fbDBRecordArraySelect(bExecute:=FALSE);
             IF NOT bError_ReadEmployee THEN
                 nState        :=0;
             ELSE
```

```
                nState          :=255;
            END_IFEND_IF
    255://Error State
        ;
END_CASE
```

Mit einer positiven Flanke an der Variable "bStart" werden die XPath-Kommandos abgesetzt und die einzelnen Elemente aus der XML-Datei gelesen. Die Ergebnisse stehen in den Variablen "sFactoryName", "stFactoryInfo" und "aEmployees".

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

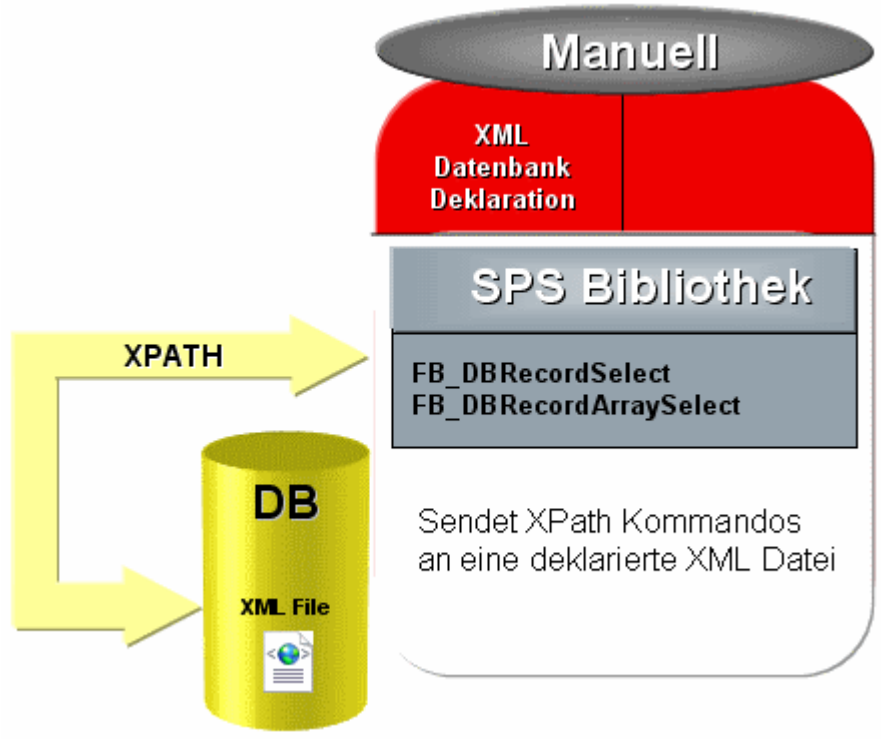
7.2.8 XPath-Beispiel mit XML-Schema

Mithilfe des Funktionsbausteins FB_DBRecordSelect oder FB_DBRecordArraySelect können XPath-Kommandos abgesetzt 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: https://infosys.beckhoff.com/content/1031/TF6420_Tc3_Database_Server/Resources/3494041099.zip



Verwendeter Datenbanktyp	XML
Kompatible Datenbanktypen	XML
Verwendete Funktionsbausteine	FB_DBRecordSelect
Einzubindende Bibliotheken	Tc2_Database, Tc2_System, Tc2_Standard, Tc2_Utilities
Download Dateiliste	TcDBSrv_InfoSysSamples.tszip, 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
```

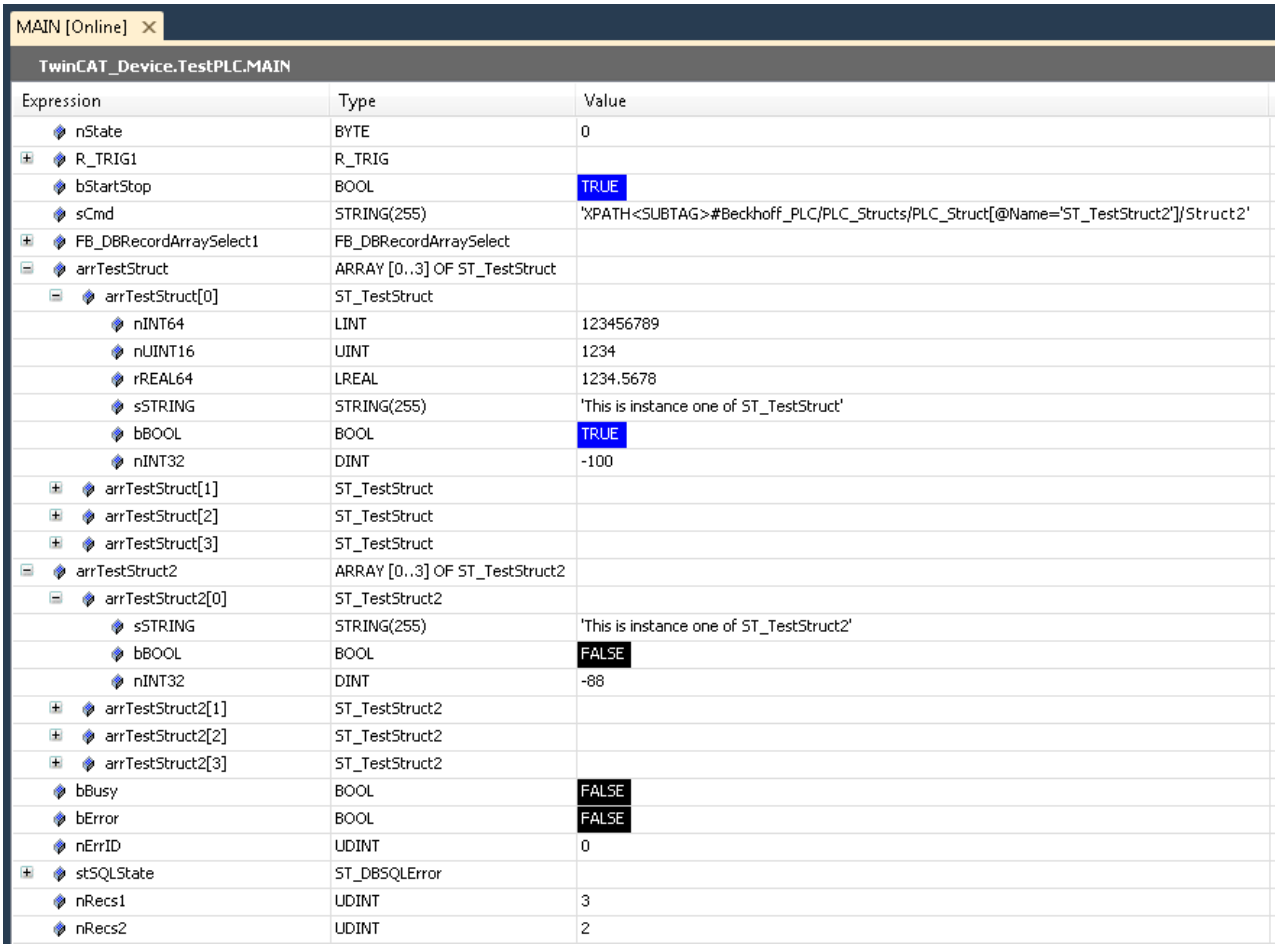
MAIN-Programm

```
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_IFEND_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_IFEND_IF
 255: (* Error Step*)
 ;
END_CASE
```

Mit einer positiven Flanke an der Toggle-Variable „bStartStop“ wird das Auslesen gestartet.



MAIN [Online]

TwinCAT_Device.TestPLC.MAIN

Expression	Type	Value
nState	BYTE	0
R_TRIG1	R_TRIG	
bStartStop	BOOL	TRUE
sCmd	STRING(255)	'XPATH<SUBTAG>#Beckhoff_PLC/PLC_Structs/PLC_Struct[@Name='ST_TestStruct2']/Struct2'
FB_DBRecordArraySelect1	FB_DBRecordArraySelect	
arrTestStruct	ARRAY [0..3] OF ST_TestStruct	
arrTestStruct[0]	ST_TestStruct	
nINT64	LINT	123456789
nUINT16	UINT	1234
rREAL64	LREAL	1234.5678
sSTRING	STRING(255)	'This is instance one of ST_TestStruct'
bBOOL	BOOL	TRUE
nINT32	DINT	-100
arrTestStruct[1]	ST_TestStruct	
arrTestStruct[2]	ST_TestStruct	
arrTestStruct[3]	ST_TestStruct	
arrTestStruct2	ARRAY [0..3] OF ST_TestStruct2	
arrTestStruct2[0]	ST_TestStruct2	
sSTRING	STRING(255)	'This is instance one of ST_TestStruct2'
bBOOL	BOOL	FALSE
nINT32	DINT	-88
arrTestStruct2[1]	ST_TestStruct2	
arrTestStruct2[2]	ST_TestStruct2	
arrTestStruct2[3]	ST_TestStruct2	
bBusy	BOOL	FALSE
bError	BOOL	FALSE
nErrID	UDINT	0
stSQLState	ST_DBSQLError	
nRecs1	UDINT	3
nRecs2	UDINT	2

Voraussetzungen

Entwicklungsumgebung	Zielplattform	Einzubindende SPS Bibliotheken
TwinCAT v3.0.0	PC oder CX (x86)	Tc2_Database

8 Anhang

8.1 Rückgabewerte

8.1.1 Tc3_Database

8.1.1.1 SPS-Rückgabewerte

Die Fehlerausgabe an allen SPS-Bausteinen der Tc3_Database.compiled-Library wird mithilfe des I_TcResultEvent-Interfaces aus der Tc3_Eventlogger.compiled-Library durchgeführt. Eine ausführliche Beschreibung des aufgetretenen Events wie auch eine Klassifizierung ist durch diese neue Interface-Struktur möglich.

```
Interface I_TcMessage [▶ 228]
    nEventId: UDINT;
    EventClass: GUID;
    eSeverity: TcEventSeverity [▶ 229];
    ipSourceInfo: I_TcSourceInfo;
```

nEventID: Spezifischer Event Code

EventClass: GUID

EventClassName: Entsprechende Event-Class-Bezeichnung kann mit der Methode RequestEventClassName ausgelesen werden

eSeverity: Einstufung des Events: von „Info“ bis „Kritischer Fehler“

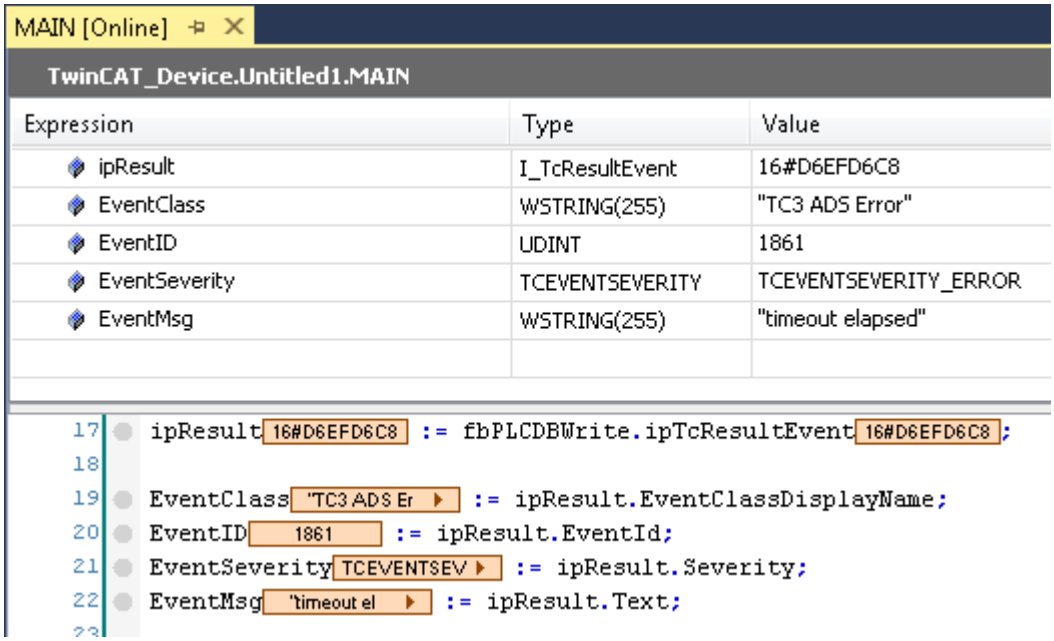
ipSourceInfo: Pfad, der den Ort des aufgetretenen Events wiedergibt.

Text: Beschreibung des Events im „Klar Text“ kann mit der Methode RequestEventText ausgelesen werden

Folgende Event-Klassen können auftreten:

- **TC3 ADS Error**
  ADS-Fehler, die bei der Kommunikation zum TwinCAT Database Server auftreten können.
- **TC3 Database Server Internal Error**
  Interne Fehler, die bei falscher Konfiguration des TwinCAT Database Servers auftreten können.
- **TC3 Database Server Database Error**
  Datenbankfehler, die bei der Kommunikation mit den entsprechenden Datenbanken auftreten können. Die verschiedenen Datenbank-spezifischen Fehlercodes werden auf eine Datenbankfehlerliste gemappt. Die Datenbank spezifischen Codes werden bei Bedarf ins ErrorLog geschrieben.
- **TC3 Database Server ADS Device Error**
  ADS-Fehler, die bei der internen Kommunikation mit konfigurierten ADS-Geräten auftreten können.
- **TC3 Database Server NoSQL Error**
  Datenbankfehler einer NoSQL Datenbank, der bei der Kommunikation mit den entsprechenden Datenbanken aufgetreten ist.

8.1.1.2 ADS-Rückgabe-Codes



Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet, nicht erreichbar oder nicht installiert.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Router Fehlercodes

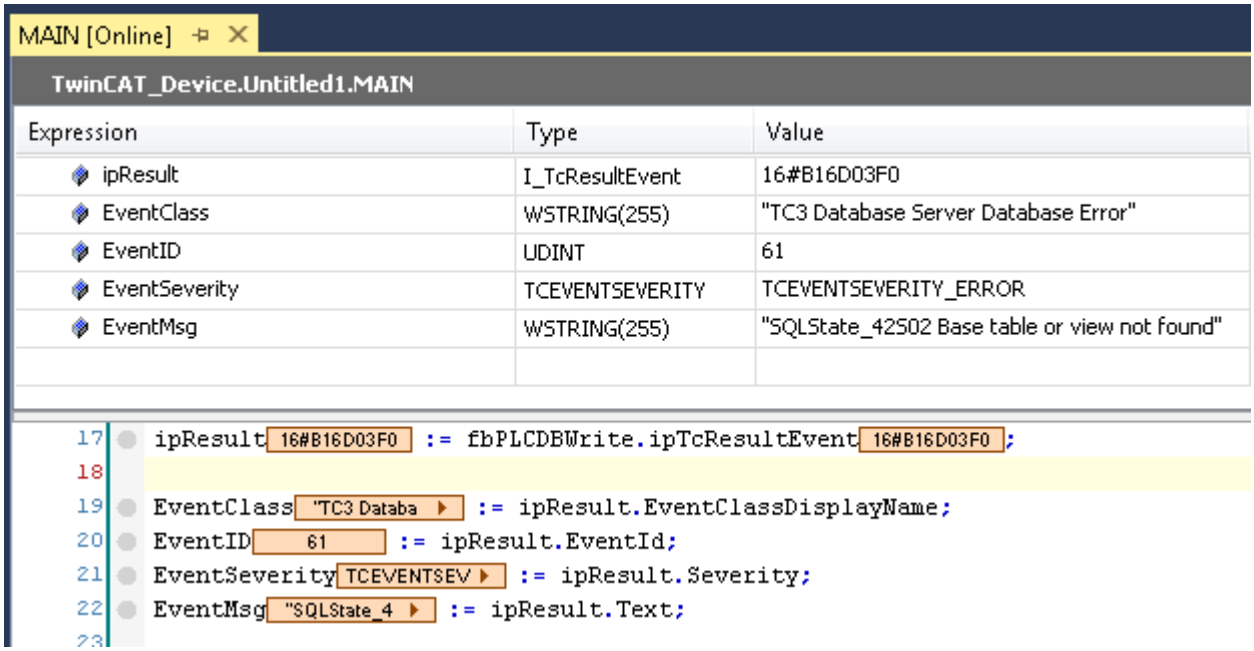
Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet. Mehrere Ursachen sind möglich. Beispielsweise beim Anlegen von Routen, dass ein falsches Passwort angegeben wurde.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert. Mehrere Ursachen sind möglich. Beispielsweise, dass eine Unidirectionale ADS Route in die umgekehrte Richtung verwendet wird.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.

Hex	Dec	HRESULT	Name	Beschreibung
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

8.1.1.3 Datenbank Rückgabe-Codes



ErrorCode	ErrorName	ErrorDescription
1	DB_SQLState_01000	General warning
2	DB_SQLState_01001	Cursor operation conflict
3	DB_SQLState_01002	Disconnect error
4	DB_SQLState_01003	NULL value eliminated in set function
5	DB_SQLState_01004	String data, right-truncated
6	DB_SQLState_01006	Privilege not revoked
7	DB_SQLState_01007	Privilege not granted
8	DB_SQLState_01S00	Invalid connection string attribute
9	DB_SQLState_01S01	Error in row
10	DB_SQLState_01S02	Option value changed
11	DB_SQLState_01S06	Attempt to fetch before the result set returned the first rowset
12	DB_SQLState_01S07	Fractional truncation
13	DB_SQLState_01S08	Error saving File DSN
14	DB_SQLState_01S09	Invalid keyword
15	DB_SQLState_07000	Dynamic SQL error
16	DB_SQLState_07001	Wrong number of parameters
17	DB_SQLState_07002	COUNT field incorrect
18	DB_SQLState_07005	Prepared statement not a cursor-specification
19	DB_SQLState_07006	Restricted data type attribute violation
20	DB_SQLState_07009	Invalid descriptor index
21	DB_SQLState_07S01	Invalid use of default parameter
22	DB_SQLState_08000	Connection exception
23	DB_SQLState_08001	Client unable to establish connection
24	DB_SQLState_08002	Connection name in use
25	DB_SQLState_08003	Connection not open
26	DB_SQLState_08004	Server rejected the connection
27	DB_SQLState_08007	Connection failure during transaction
28	DB_SQLState_08S01	Communication link failure
29	DB_SQLState_21000	Cardinality violation
30	DB_SQLState_21S01	Insert value list does not match column list
31	DB_SQLState_21S02	Degree of derived table does not match column list
32	DB_SQLState_22000	Data exception
33	DB_SQLState_22001	String data, right-truncated
34	DB_SQLState_22002	Indicator variable required but not supplied
35	DB_SQLState_22003	Numeric value out of range
36	DB_SQLState_22007	Invalid datetime format
37	DB_SQLState_22008	Datetime field overflow
38	DB_SQLState_22012	Division by zero
39	DB_SQLState_22015	Interval field overflow
40	DB_SQLState_22018	Invalid character value for cast specification
41	DB_SQLState_22019	Invalid escape character
42	DB_SQLState_22025	Invalid escape sequence
43	DB_SQLState_22026	String data, length mismatch

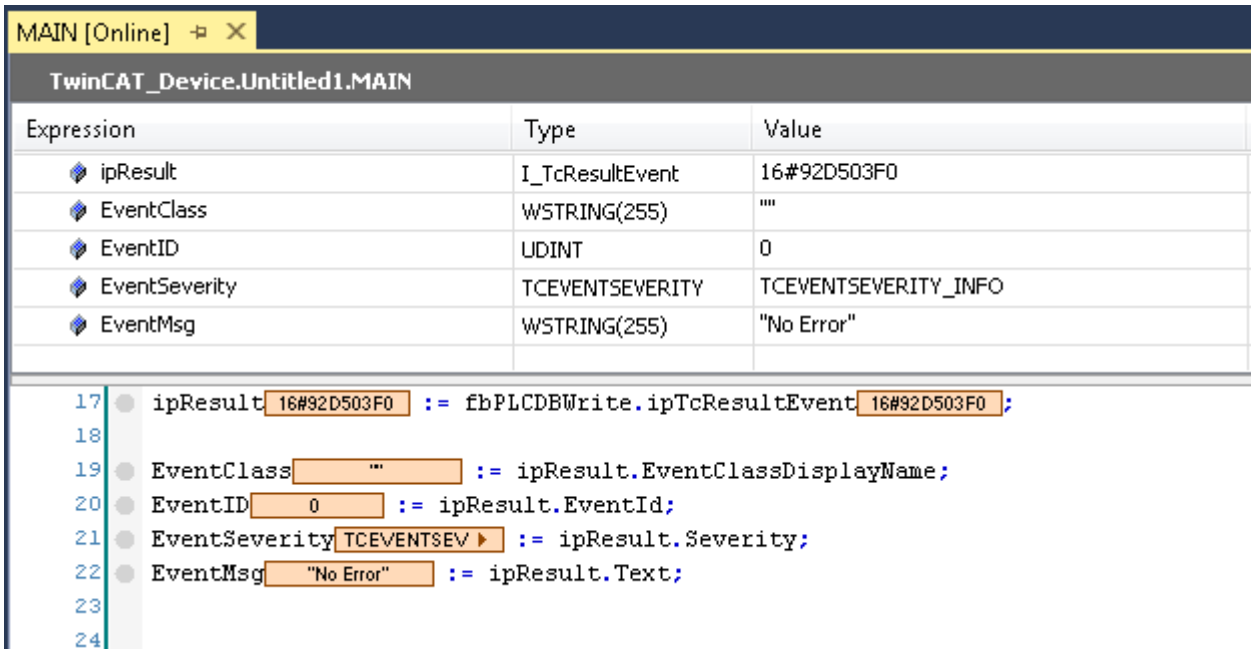
44	DB_SQLState_23000	Integrity constraint violation
45	DB_SQLState_24000	Invalid cursor state
46	DB_SQLState_25000	Invalid transaction state
47	DB_SQLState_25S01	Transaction state
48	DB_SQLState_25S02	Transaction is still active
49	DB_SQLState_25S03	Transaction is rolled back
50	DB_SQLState_28000	Invalid authorization specification
51	DB_SQLState_34000	Invalid cursor name
52	DB_SQLState_3C000	Duplicate cursor name
53	DB_SQLState_3D000	Invalid catalog name
54	DB_SQLState_3F000	Invalid schema name
55	DB_SQLState_40000	Transaction rollback
56	DB_SQLState_40001	Serialization failure
57	DB_SQLState_40002	Integrity constraint violation
58	DB_SQLState_40003	Statement completion unknown
59	DB_SQLState_42000	Syntax error or access violation
60	DB_SQLState_42S01	Base table or view already exists
61	DB_SQLState_42S02	Base table or view not found
62	DB_SQLState_42S11	Index already exists
63	DB_SQLState_42S12	Index not found
64	DB_SQLState_42S21	Column already exists
65	DB_SQLState_42S22	Column not found
66	DB_SQLState_44000	WITH CHECK OPTION violation
67	DB_SQLState_HY000	General error
68	DB_SQLState_HY001	Memory allocation error
69	DB_SQLState_HY003	Invalid application buffer type
70	DB_SQLState_HY004	Invalid SQL data type
71	DB_SQLState_HY007	Associated statement is not prepared
72	DB_SQLState_HY008	Operation canceled
73	DB_SQLState_HY009	Invalid use of null pointer
74	DB_SQLState_HY010	Function sequence error
75	DB_SQLState_HY011	Attribute cannot be set now
76	DB_SQLState_HY012	Invalid transaction operation code
77	DB_SQLState_HY013	Memory management error
78	DB_SQLState_HY014	Limit on the number of handles exceeded
79	DB_SQLState_HY015	No cursor name available
80	DB_SQLState_HY016	Cannot modify an implementation row descriptor
81	DB_SQLState_HY017	Invalid use of an automatically allocated descriptor handle

82	DB_SQLState_HY018	Server declined cancel request
83	DB_SQLState_HY019	Non-character and non-binary data sent in pieces
84	DB_SQLState_HY020	Attempt to concatenate a null value
85	DB_SQLState_HY021	Inconsistent descriptor information
86	DB_SQLState_HY024	Invalid attribute value
87	DB_SQLState_HY090	Invalid string or buffer length
88	DB_SQLState_HY091	Invalid descriptor field identifier
89	DB_SQLState_HY092	Invalid attribute/option identifier
90	DB_SQLState_HY095	Function type out of range
91	DB_SQLState_HY096	Invalid information type
92	DB_SQLState_HY097	Column type out of range
93	DB_SQLState_HY098	Scope type out of range
94	DB_SQLState_HY099	Nullable type out of range
95	DB_SQLState_HY100	Uniqueness option type out of range
96	DB_SQLState_HY101	Accuracy option type out of range
97	DB_SQLState_HY103	Invalid retrieval code
98	DB_SQLState_HY104	Invalid precision or scale value
99	DB_SQLState_HY105	Invalid parameter type
100	DB_SQLState_HY106	Fetch type out of range
101	DB_SQLState_HY107	Row value out of range
102	DB_SQLState_HY109	Invalid cursor position
103	DB_SQLState_HY110	Invalid driver completion
104	DB_SQLState_HY111	Invalid bookmark value
105	DB_SQLState_HYC00	Optional feature not implemented
106	DB_SQLState_HYT00	Timeout expired
107	DB_SQLState_HYT01	Connection timeout expired
108	DB_SQLState_IM001	Driver does not support this function
109	DB_SQLState_IM002	Data source name not found and no default driver specified
110	DB_SQLState_IM003	Specified driver could not be loaded
111	DB_SQLState_IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed
112	DB_SQLState_IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed
113	DB_SQLState_IM006	Driver's SQLSetConnectAttr failed
114	DB_SQLState_IM007	No data source or driver specified dialog prohibited
115	DB_SQLState_IM008	Dialog failed
116	DB_SQLState_IM009	Unable to load translation DLL
117	DB_SQLState_IM010	Data source name too long
118	DB_SQLState_IM011	Driver name too long
119	DB_SQLState_IM012	DRIVER keyword syntax error
120	DB_SQLState_IM013	Trace file error
121	DB_SQLState_IM014	Invalid name of File DSN
122	DB_SQLState_IM015	Corrupt file data source
200	DB_FunctionNotImplemented	Function not implemented
255	DB_SQLState_Unspecified	Unspecified Error

8.1.1.4 NoSQL Datenbank Rückgabe-Codes

ErrorCode	ErrorName	ErrorDescription
0	NoSql_0	No Error
1	NoSql_Specific_1	Internal MongoDB exception. Please check the information log for more details.
2	NoSql_Specific_2	Database side error. Please check the information log for more details.
8	NoSql_Specific_8	Timeout error occurred on command execution
9	NoSql_Specific_9	Format error occurred on command execution. Please check the command or document syntax.
11	NoSql_Specific_11	RunCommand execution failed. Please check the event log for more information
12	NoSql_Specific_12	There is no metadata including the tableschema for this table
13	NoSql_Specific_13	Syntax error in command / filter
14	NoSql_Specific_14	Connection error occured
15	NoSql_Specific_15	Authentication failed
16	NoSql_Specific_16	Error occured during write operation
20	NoSql_Specific_20	Functionality is not supported by the TwinCAT Database Server
48	NoSql_Specific_48	Namespace / CollectionName already exists
51	NoSql_Specific_51	The queried data is empty
141	NoSql_Specific_141	Internal TwinCAT 3 Database Server Error

8.1.1.5 TwinCAT Database Server Codes



ErrorCode	ErrorName	ErrorDescription
0	InternalErrorCode_0	No Error
1	InternalErrorCode_1	NULL Values not allowed
2	InternalErrorCode_2	FB_DBRead selected value is NULL
3	InternalErrorCode_3	DBID is unknown
4	InternalErrorCode_4	ADSDevID is unknown
5	InternalErrorCode_5	No open database connection found for DBID: xy
6	InternalErrorCode_6	No open ADS Device connection found for ADSDevID: xy
7	InternalErrorCode_7	Init of AutoLog Groups failed
8	InternalErrorCode_8	AutoLog could NOT be started. TwinCAT Database Server has no valid device state
9	InternalErrorCode_9	Record select value return => Wrong Parametersize from ADS-Device
10	InternalErrorCode_10	Invalid Parameter
11	InternalErrorCode_11	Couldn't find PLCDBWrite Value in list
12	InternalErrorCode_12	No valid symbol TYPE
13	InternalErrorCode_13	Invalid parameter size
14	InternalErrorCode_14	Execution timeout
15	InternalErrorCode_15	Database connection already open
16	InternalErrorCode_16	Database connection not initialized
1000	InternalErrorCode_1000	Unknown internal Error

8.1.2 Tc2_Database

8.1.2.1 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 401]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 402]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 403]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 405]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet, nicht erreichbar oder nicht installiert.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet. Mehrere Ursachen sind möglich. Beispielsweise beim Anlegen von Routen, dass ein falsches Passwort angegeben wurde.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert. Mehrere Ursachen sind möglich. Beispielsweise, dass eine Unidirectionale ADS Route in die umgekehrte Richtung verwendet wird.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.

Hex	Dec	HRESULT	Name	Beschreibung
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	Die Anfrage wurde abgebrochen.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

8.1.2.2 Fehlercode - Übersicht TwinCAT Database Server

Code (Hex)	Code (Dez)	Beschreibung
0x0001 + ADS-FehlerCode	65537 - 131071	ADS-Fehlercode vom Deklarierten ADS-Gerät
0x00020001	131073	Microsoft SQL Compact Datenbank (Fehlercode)
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 [▶ 413]
0x04000001	67108865	XML Datenbank (Fehlercode)
0x08000001	134217729	ASCII Datenbank (Fehlercode)

Wird einer der oben genannten Fehlercodes am "nErrID"-Ausgang eines Funktionsbausteins ausgegeben, ist ein Fehler beim Ausführen eines SQL-Statements aufgetreten. Der SQL-Fehlercode wird dann am "sSQLState"-Ausgang des Funktionsbausteins ausgegeben. Der "sSQLState"-Ausgang ist von Datentyp ST_DBSQLError [▶ 326]. 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 / _OleDB_Errorcodes.htm [▶ 408]
Microsoft SQL Datenbank	OleDB_Errorcodes.htm [▶ 408]
Microsoft Access Datenbank	OleDB_Errorcodes.htm [▶ 408]
MySQL Datenbank	https://dev.mysql.com/doc/mysql-errors/8.0/en/client-error-reference.html
Oracle Datenbank	https://docs.oracle.com/cd/E11882_01/server.112/e17766/toc.htm
DB2 Datenbank	https://www.ibm.com/docs/en/db2-for-zos/11?topic=codes-sql-error
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 [▶ 413]
ASCII Datenbank	TcDBServer_ASCII_Errorcodes.htm [▶ 413]

8.1.2.3 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.
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.

Wert	Beschreibung
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.
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.

Wert	Beschreibung
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.
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.

Wert	Beschreibung
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.
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.

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

8.1.2.4 ASCII-Fehlercodes

Wert	Beschreibung
1	Funktion nicht verfügbar
2	Syntaxfehler
3	Datei konnte nicht geöffnet werden.

8.1.2.5 XML-Fehlercodes

Wert	Beschreibung
1	Funktion nicht verfügbar
2	XML-Datei konnte nicht geladen werden
3	XML-Schema konnte nicht geladen werden
4	Syntaxfehler
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

8.1.2.6 Interne Fehlercodes

Fehlercode	Beschreibung
0x02000001	NULL values are not allowed
0x02000002	FB_DBRead selected value is NULL
0x02000003	DBID is unknown
0x02000004	ADSDevID is unknown
0x02000005	No open database connection found for DBID xy
0x02000006	No open ADS device connection found for ADSDevID xy

8.2 FAQ - Häufig gestellte Fragen und Antworten

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

1. Welche Performance lässt sich mit dem TwinCAT Database Server erzielen? [▶ 414]

2. Werden so genannte Stored Procedures unterstützt? [▶ 414]

3. Welche Datenbanktypen werden von dem TwinCAT Database Server unterstützt? [▶ 414]

4. Können alte Database Server Konfigurationen noch in aktuellen Versionen genutzt werden? [▶ 414]

5. Wie können einzelne Variablen in eine bereits vorhandene Datenbankstruktur geschrieben bzw. aus ihr heraus gelesen werden? [▶ 414]

6. Können mehrere Datensätze gleichzeitig in eine DB geschrieben werden? [▶ 414]

7. Wie kann der TwinCAT Database Server im Netzwerk betrieben werden? [▶ 414]

8. Welche Funktionen des TwinCAT Database Servers werden vom Datenbanktyp "XML" unterstützt? [▶ 415]

9. Welche Visual Studio Versionen werden aktuell vom Database Server Konfigurator unterstützt? [▶ 415]

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 von 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. Der Funktionsbaustein FB_SQLStoredProcedure [▶ 205] ist dafür in der SPS-Bibliothek vorgesehen. Auch im sogenannten SQL Query Editor können Stored Procedures getestet werden und es kann sogar entsprechender SPS-Code für den Funktionsbaustein FB_SQLStoredProcedure erzeugt werden. Diese Funktion wird nicht von jeder Datenbank unterstützt.

Welche Datenbanktypen werden von dem TwinCAT Database Server unterstützt?
Informationen zu den unterstützten Datenbanken finden Sie im Abschnitt "Datenbanken [▶ 126]".

Können alte Database Server Konfigurationen noch in aktuellen Versionen genutzt werden?
Selbstverständlich sorgen wir für eine entsprechende Kompatibilität. Dies haben wir auch bei großen Versionssprüngen bzw. bei einem vollständigen Redesign (alt: 3.**0**.x, neu: 3.**1**.x) entsprechend beachtet. Details dazu erfahren Sie im Abschnitt "Kompatibilität [▶ 21]".

Wie können einzelne Variablen in eine bereits vorhandene Datenbankstruktur geschrieben bzw. aus ihr heraus gelesen?
Mithilfe des Funktionsbausteins FB_SQLCommand [▶ 200] können einzelne Variablen in eine vorhandene Datenbankstruktur geschrieben oder aus ihr ausgelesen werden.

Können mehrere Datensätze gleichzeitig in eine DB geschrieben werden?
Dies ist abhängig von der verwendeten Datenbank. Mit einer Microsoft SQL Datenbank wäre dies in Verbindung mit dem Funktionsbaustein FB_SQLCommand [▶ 200] möglich, da mehrere SQL Insert-Befehle, getrennt durch ein Semikolon, 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. Nähere Informationen zu unterstützten Netzwerk-Topologien finden Sie im Abschnitt "Einsatzgebiete und Netzwerktopologien [▶ 20]".

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 Informationen finden Sie im Abschnitt "XML-Database [▶ 138]".

Welche Visual Studio Versionen werden aktuell vom Database Server Konfigurator unterstützt?

Aktuell werden die Visual Studio® Versionen 2013, 2015 und 2017 mit unserer Konfigurator-Integration [▶ 24] unterstützt.

8.3 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

Mehr Informationen:
www.beckhoff.com/tf6420

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