Manual

TC3 Database Server

TwinCAT 3

Version: 1.9
Date: 2019-04-24
Order No.: TF6420

BECKHOFF

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with the applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, EtherCAT®, Safety over EtherCAT®, TwinSAFE®, XFC® and XTS® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, DE102004044764, DE102007017835
with corresponding applications or registrations in various other countries.

The TwinCAT Technology is covered, including but not limited to the following patent applications and patents:
EP0851348, US6167425 with corresponding applications or registrations in various other countries.

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 Safety instructions

Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

ℹ **Tip or pointer**

This symbol indicates information that contributes to better understanding.

2 Overview

The TwinCAT Database Server enables data exchange between the TwinCAT system and various database systems. For small applications it can be used via a configurator, without intervention in the existing program code. For complex tasks the Database Server offers a large library of PLC function blocks for maximum flexibility. SQL commands such as Insert or Select can be used directly from the PLC, for example. To take load off the PLC, if required, procedures can be stored (Stored Procedures) and then called up from the databases. In this case the parameters transferred by the corresponding PLC function block are used by the database in conjunction with the Stored Procedure, and results can be returned to the controller.

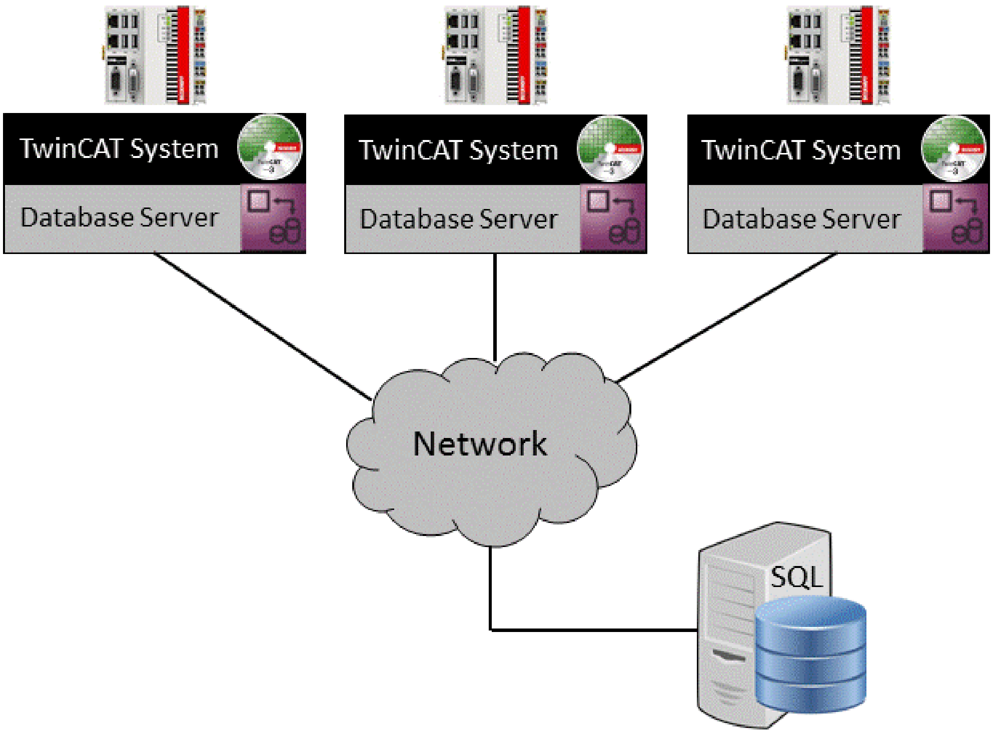
The TwinCAT Database Server supports a wide range or different database systems, MS SQL, MS SQL Compact, MS Access, MySQL, PostgreSQL, DB2, Oracle, Interbase, Firebird, ASCII (e.g. .txt or .csv) and XML files, now also including NoSQL databases, based on support of MongoDB. (See also: Declaration of the different database types)

Components

- TwinCAT Database Server [▶ 19]: The service is started and stopped together with TwinCAT. It forms the link between the TwinCAT system and the database.
- Configurator [▶ 96]: The TwinCAT Database Server Configurator facilitates visual setting of the database parameters required for basic communication with the respective database.
- PLC library [▶ 280]: The PLC library includes various function blocks. They enable establishment of a database connection, creation of a new table, writing of data into any table structure using Insert commands, and reading via Select commands. It is also possible to update or delete database entries and trigger stored procedures. NoSQL databases have their own function blocks that are optimized for handling flexible JSON documents in the PLC, for example. The principle of operation is identical.

Principle of operation

Within the TwinCAT system the Database Server communicates via ADS. Externally it links to the respective configured database. Possible network topologies can be found in section "Areas of application and network technologies [▶ 20]".



3 Installation

3.1 System requirements

Technical data	TF6420 TwinCAT 3 Database Server
Target system	Windows 7, Windows 8, Windows 10, WinCE PC (x86, x64 und ARM)
.NET Framework	.Net 4.0 or higher WinCE: .NET 3.5
Min. TwinCAT version	3.1.4018
Min. TwinCAT level	TC1200 TC3 | PLC

3.2 Installation

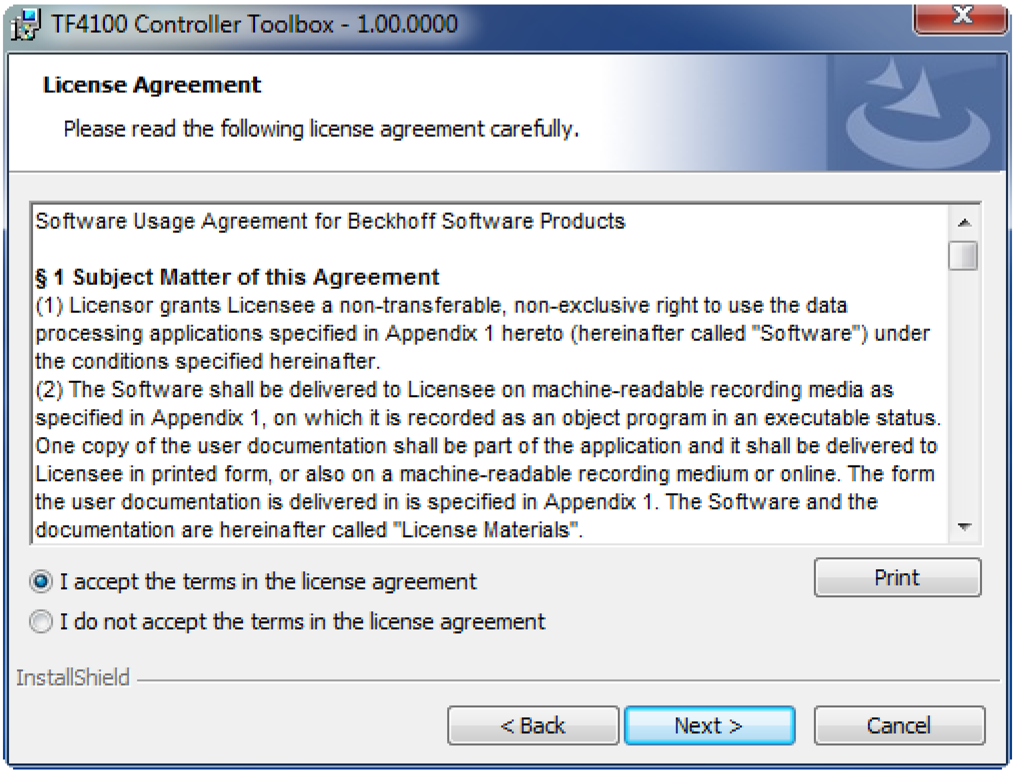
The following section describes how to install the TwinCAT 3 Function for Windows-based operating systems.

✓ The TwinCAT 3 Function setup file was downloaded from the Beckhoff website.

1. Run the setup file as administrator. To do this, select the command **Run as administrator** in the context menu of the file.

   ⇨ The installation dialog opens.

2. Accept the end user licensing agreement and click **Next**.



TF4100 Controller Toolbox - 1.00.0000
License Agreement
Please read the following license agreement carefully.

Software Usage Agreement for Beckhoff Software Products

§ 1 Subject Matter of this Agreement

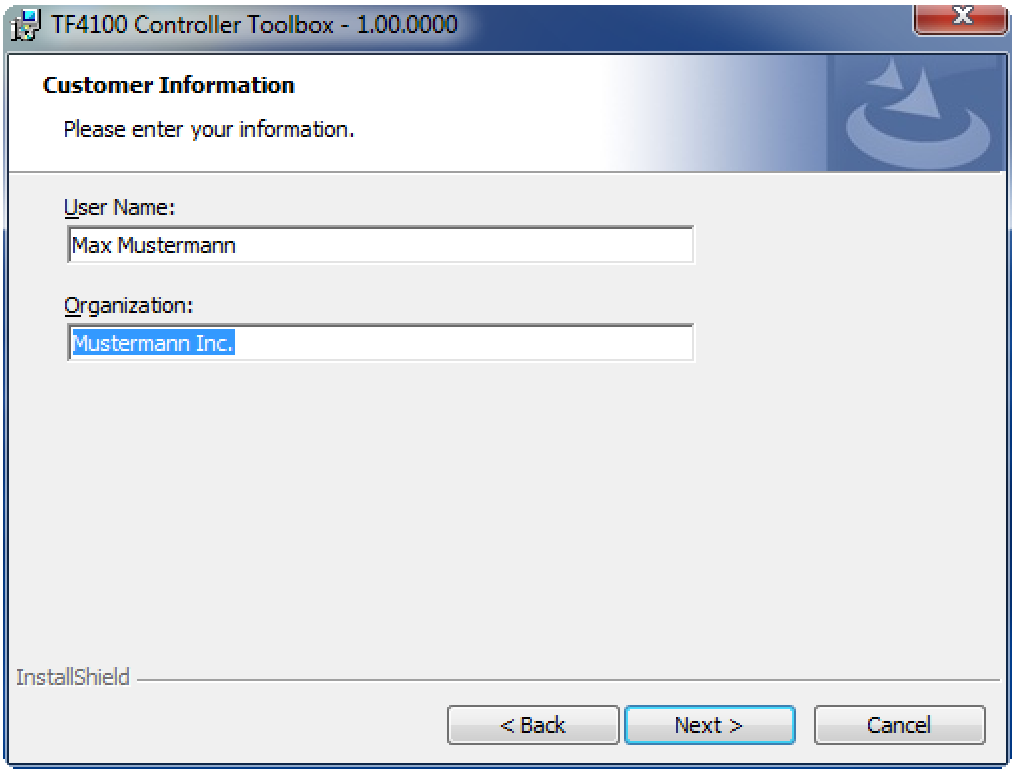
(1) Licensor grants Licensee a non-transferable, non-exclusive right to use the data processing applications specified in Appendix 1 hereto (hereinafter called "Software") under the conditions specified hereinafter.

(2) The Software shall be delivered to Licensee on machine-readable recording media as specified in Appendix 1, on which it is recorded as an object program in an executable status. One copy of the user documentation shall be part of the application and it shall be delivered to Licensee in printed form, or also on a machine-readable recording medium or online. The form the user documentation is delivered in is specified in Appendix 1. The Software and the documentation are hereinafter called "License Materials".

● I accept the terms in the license agreement

○ I do not accept the terms in the license agreement

3. Enter your user data.



TF4100 Controller Toolbox - 1.00.0000

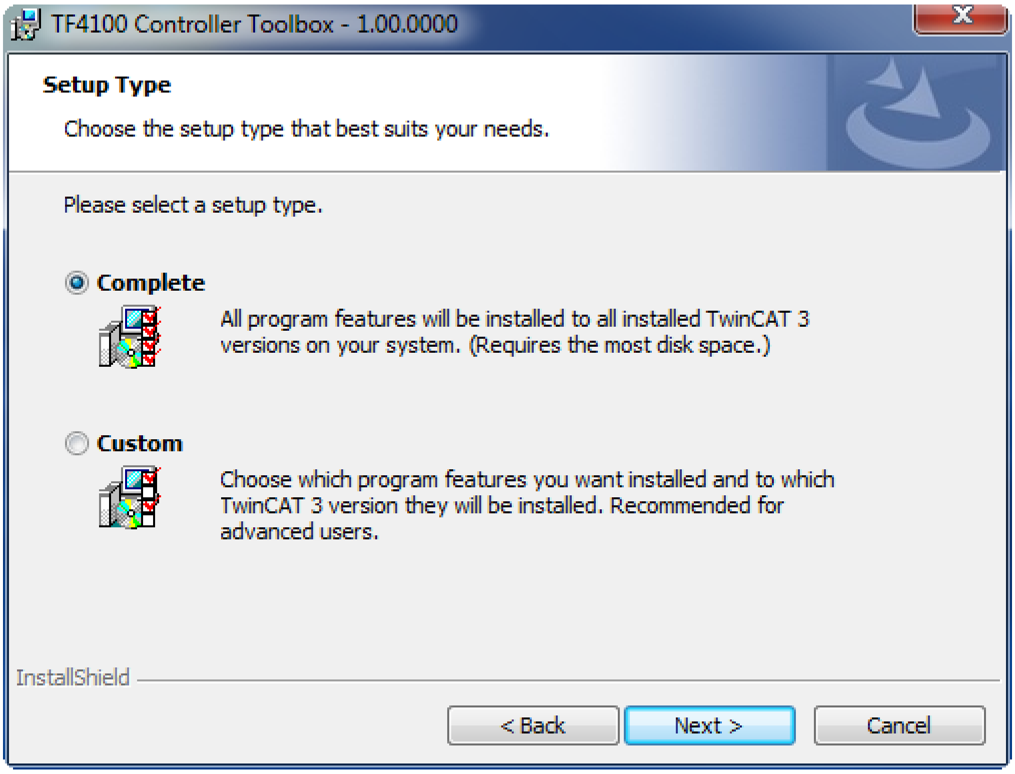
Customer Information

Please enter your information.

User Name: Max Mustermann

Organization: Mustermann Inc.

4. If you want to install the full version of the TwinCAT 3 Function, select **Complete** as installation type. If you want to install the TwinCAT 3 Function components separately, select **Custom**.



TF4100 Controller Toolbox - 1.00.0000

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

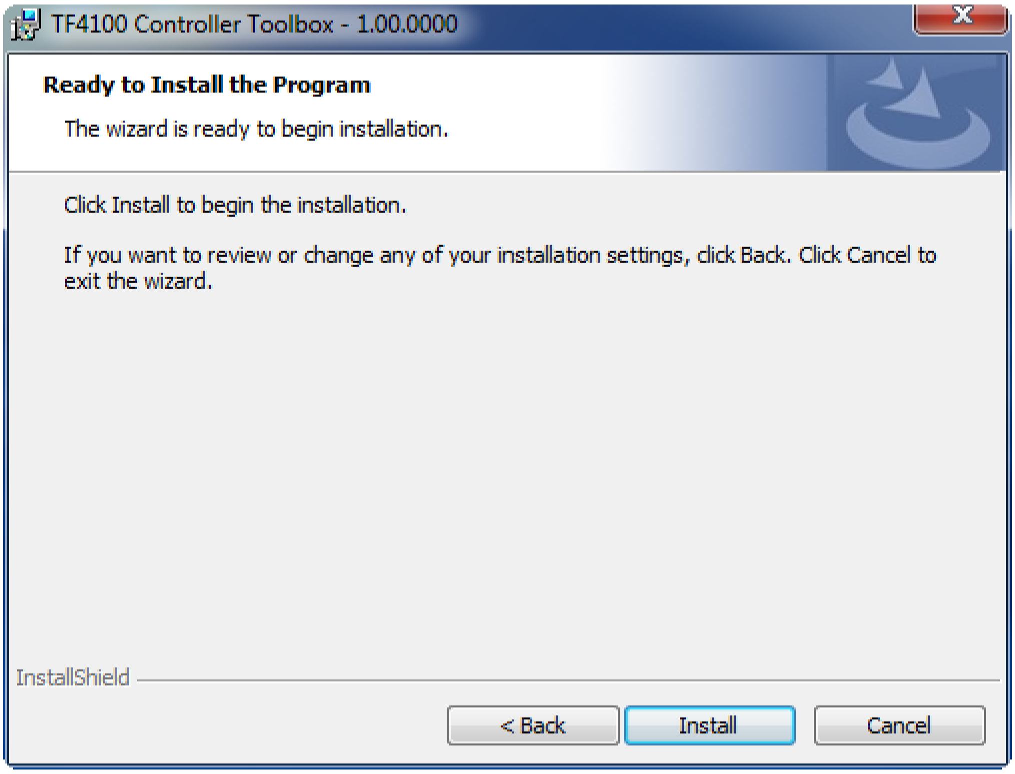
● **Complete**

All program features will be installed to all installed TwinCAT 3 versions on your system. (Requires the most disk space.)

○ **Custom**

Choose which program features you want installed and to which TwinCAT 3 version they will be installed. Recommended for advanced users.

5. Select **Next**, then **Install** to start the installation.



TF4100 Controller Toolbox - 1.00.0000

Ready to Install the Program

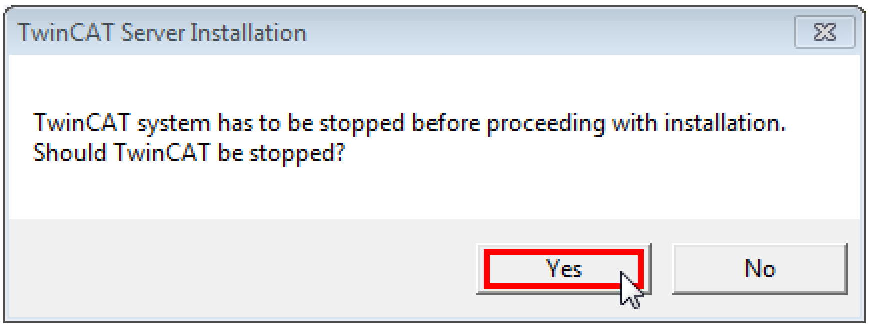
The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

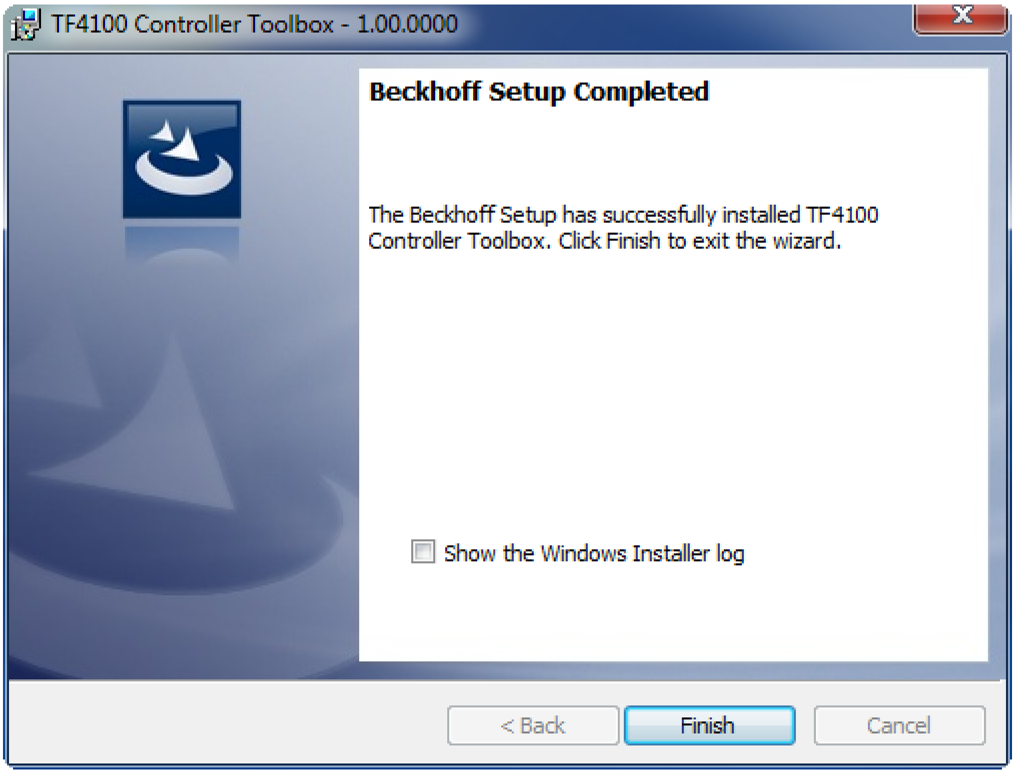
6. Confirm the dialog with **Yes**.



TwinCAT Server Installation

TwinCAT system has to be stopped before proceeding with installation.
Should TwinCAT be stopped?

7. Select **Finish** to exit the setup.



TF4100 Controller Toolbox - 1.00.0000

Beckhoff Setup Completed

The Beckhoff Setup has successfully installed TF4100 Controller Toolbox. Click Finish to exit the wizard.

□ Show the Windows Installer log

⇨ The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [▸ 11]).

3.3 Licensing

The TwinCAT 3 Function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

The licensing of a TwinCAT 3 Function is described below. The description is divided into the following sections:

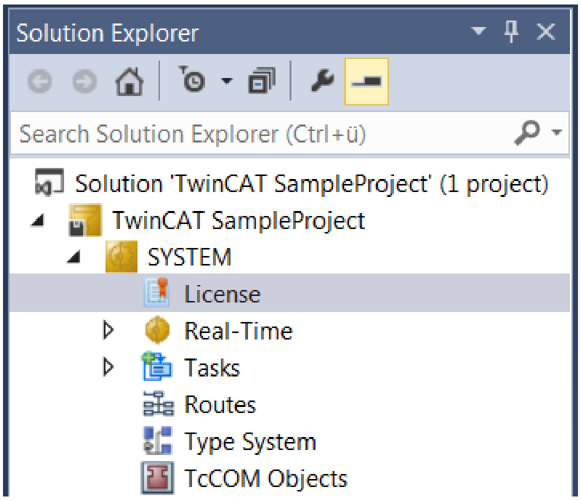
- Licensing a 7-day test version [▸ 11]
- Licensing a full version [▸ 13]

Further information on TwinCAT 3 licensing can be found in the “Licensing” documentation in the Beckhoff Information System (TwinCAT 3 > Licensing).

Licensing a 7-day test version

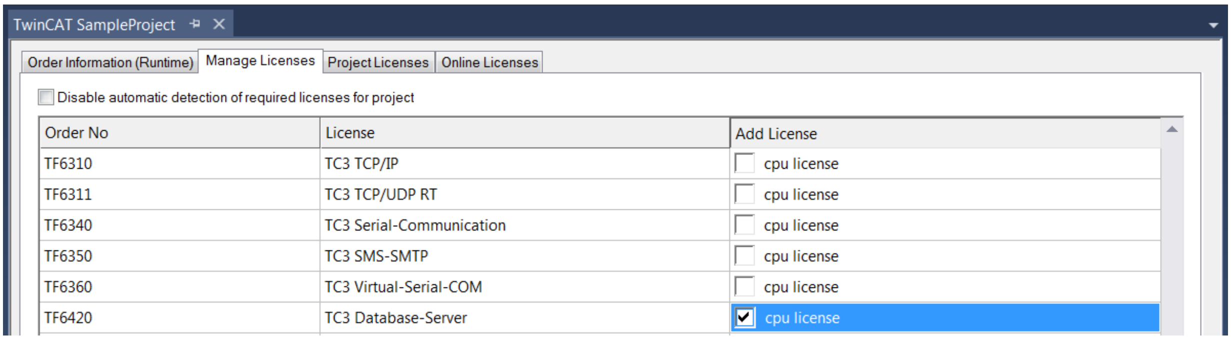
1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



⇨ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. “TF6420: TC3 Database Server“).

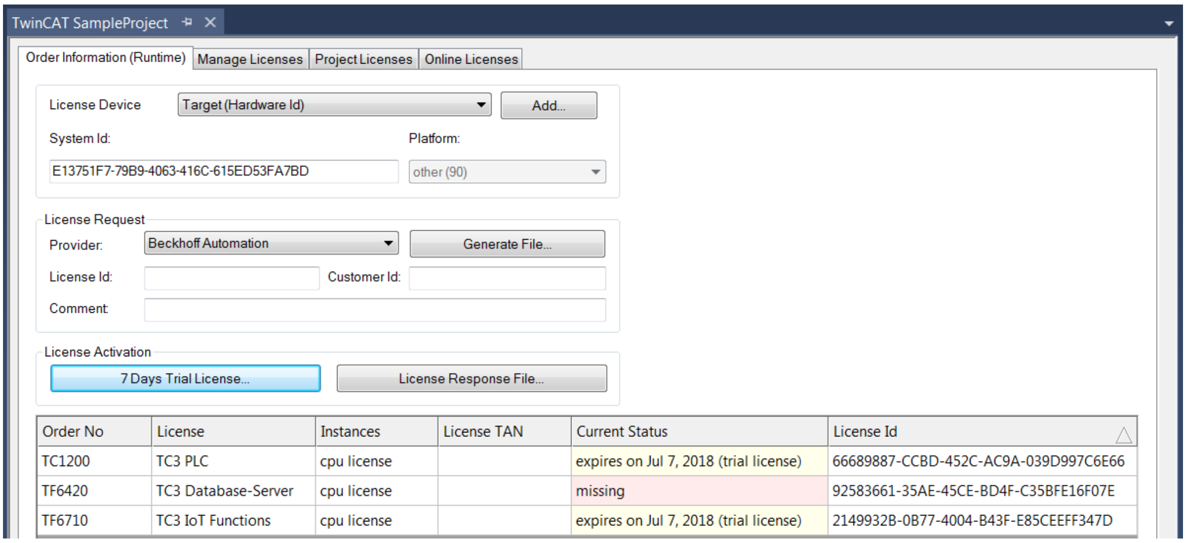


□ Disable automatic detection of required licenses for project

Order No	License	Add License
TF6310	TC3 TCP/IP	□ cpu license
TF6311	TC3 TCP/UDP RT	□ cpu license
TF6340	TC3 Serial-Communication	□ cpu license
TF6350	TC3 SMS-SMTP	□ cpu license
TF6360	TC3 Virtual-Serial-COM	□ cpu license
TF6420	TC3 Database-Server	☑ cpu license

6. Open the **Order Information (Runtime)** tab.

⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**



License Device: Target (Hardware Id)

System Id: E13751F7-79B9-4063-416C-615ED53FA7BD

Platform: other (90)

License Request

Provider: Beckhoff Automation

License Id:

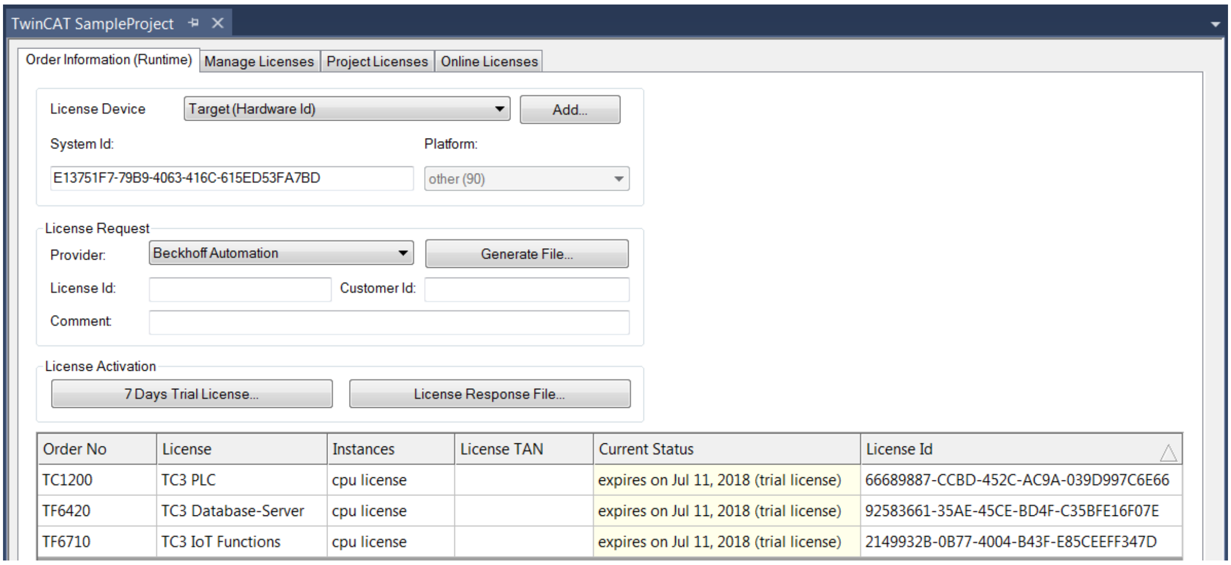
Customer Id:

Comment:

License Activation

Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 7, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TF6420	TC3 Database-Server	cpu license		missing	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 7, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

7. Click **7-Day Trial License...** to activate the 7-day trial license.



License Device: Target (Hardware Id)

System Id: E13751F7-79B9-4063-416C-615ED53FA7BD

Platform: other (90)

Provider: Beckhoff Automation

License Id:

Customer Id:

Comment:

Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 11, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TF6420	TC3 Database-Server	cpu license		expires on Jul 11, 2018 (trial license)	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 11, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.

8. Enter the code exactly as it appears, confirm it and acknowledge the subsequent dialog indicating successful activation.

   ⇨ In the tabular overview of licenses, the license status now indicates the expiration date of the license.

9. Restart the TwinCAT system.

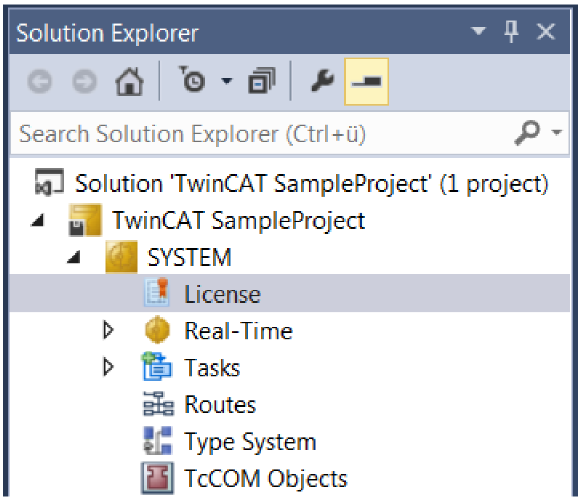
⇨ The 7-day trial version is enabled.

Licensing a full version

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.

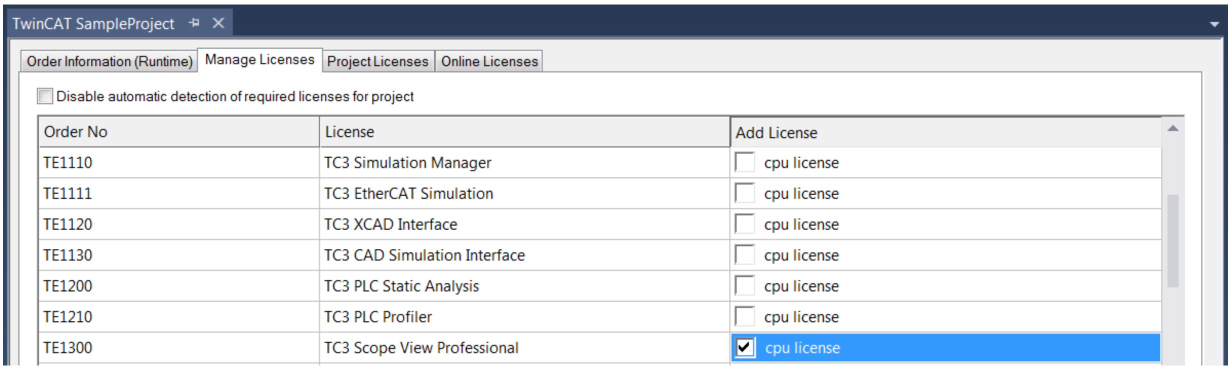
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



⇨ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. “TE1300: TC3 Scope View Professional”).

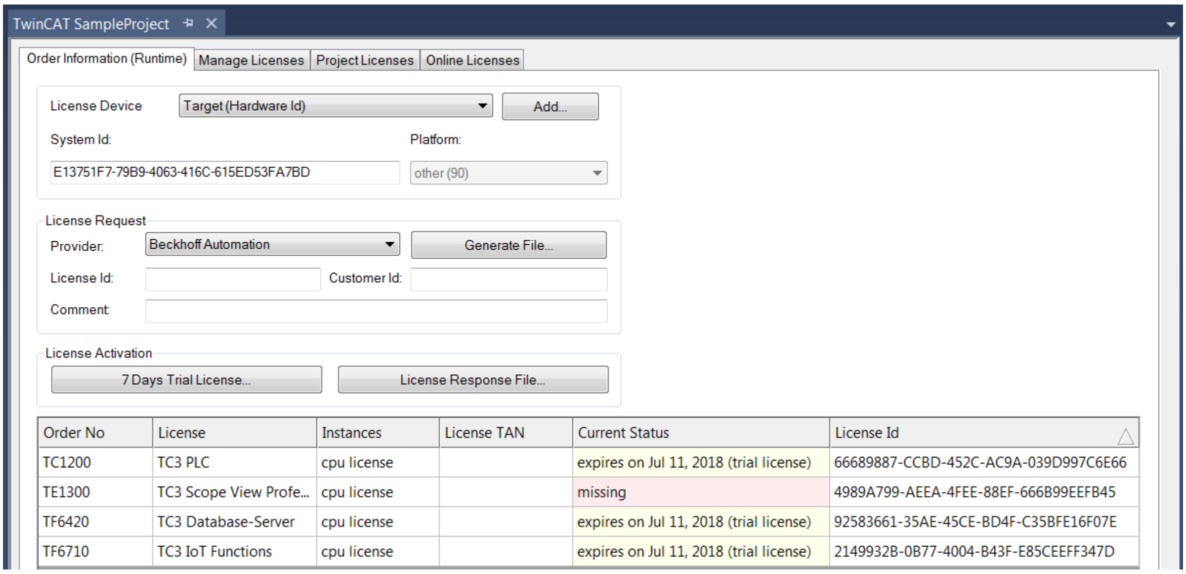


□ Disable automatic detection of required licenses for project

Order No	License	Add License
TE1110	TC3 Simulation Manager	□ cpu license
TE1111	TC3 EtherCAT Simulation	□ cpu license
TE1120	TC3 XCAD Interface	□ cpu license
TE1130	TC3 CAD Simulation Interface	□ cpu license
TE1200	TC3 PLC Static Analysis	□ cpu license
TE1210	TC3 PLC Profiler	□ cpu license
TE1300	TC3 Scope View Professional	☑ cpu license

6. Open the **Order Information** tab.

⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**



License Device: Target (Hardware Id)

System Id: E13751F7-79B9-4063-416C-615ED53FA7BD

Platform: other (90)

Provider: Beckhoff Automation

License Id:

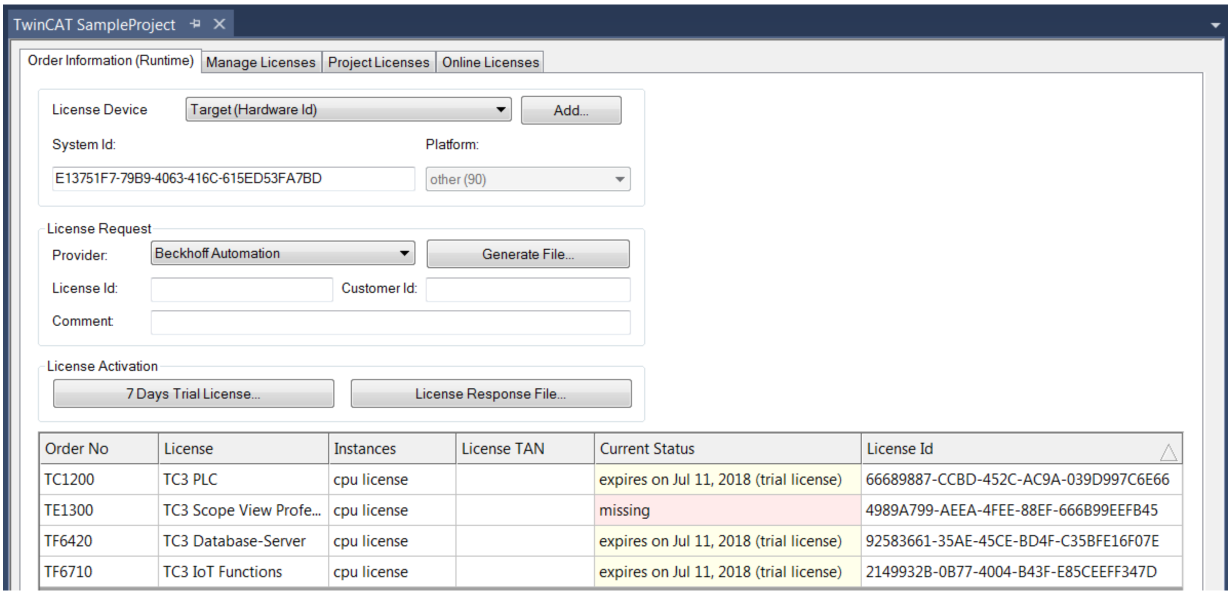
Customer Id:

Comment:

Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 11, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TE1300	TC3 Scope View Profe...	cpu license		missing	4989A799-AEEA-4FEE-88EF-666B99EEFB45
TF6420	TC3 Database-Server	cpu license		expires on Jul 11, 2018 (trial license)	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 11, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

A TwinCAT 3 license is generally linked to two indices describing the platform to be licensed:
System ID: Uniquely identifies the device
Platform level: Defines the performance of the device
The corresponding **System Id** and **Platform** fields cannot be changed.

7. Enter the order number (**License Id**) for the license to be activated and optionally a separate order number (**Customer Id**), plus an optional comment for your own purposes (**Comment**). If you do not know your Beckhoff order number, please contact your Beckhoff sales contact.



License Device: Target (Hardware Id)

System Id: E13751F7-79B9-4063-416C-615ED53FA7BD

Platform: other (90)

License Request

Provider: Beckhoff Automation

License Id:

Customer Id:

Comment:

License Activation

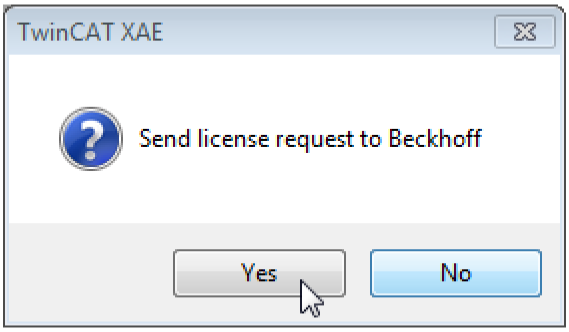
Order No	License	Instances	License TAN	Current Status	License Id
TC1200	TC3 PLC	cpu license		expires on Jul 11, 2018 (trial license)	66689887-CCBD-452C-AC9A-039D997C6E66
TE1300	TC3 Scope View Profe...	cpu license		missing	4989A799-AEEA-4FEE-88EF-666B99EEFB45
TF6420	TC3 Database-Server	cpu license		expires on Jul 11, 2018 (trial license)	92583661-35AE-45CE-BD4F-C35BFE16F07E
TF6710	TC3 IoT Functions	cpu license		expires on Jul 11, 2018 (trial license)	2149932B-0B77-4004-B43F-E85CEEFF347D

8. Click the **Generate File**... button to create a License Request File for the listed missing license.

   ⇨ A window opens, in which you can specify where the License Request File is to be stored. (We recommend accepting the default settings.)

9. Select a location and click **Save**.

   ⇨ A prompt appears asking whether you want to send the License Request File to the Beckhoff license server for verification:



Send license request to Beckhoff

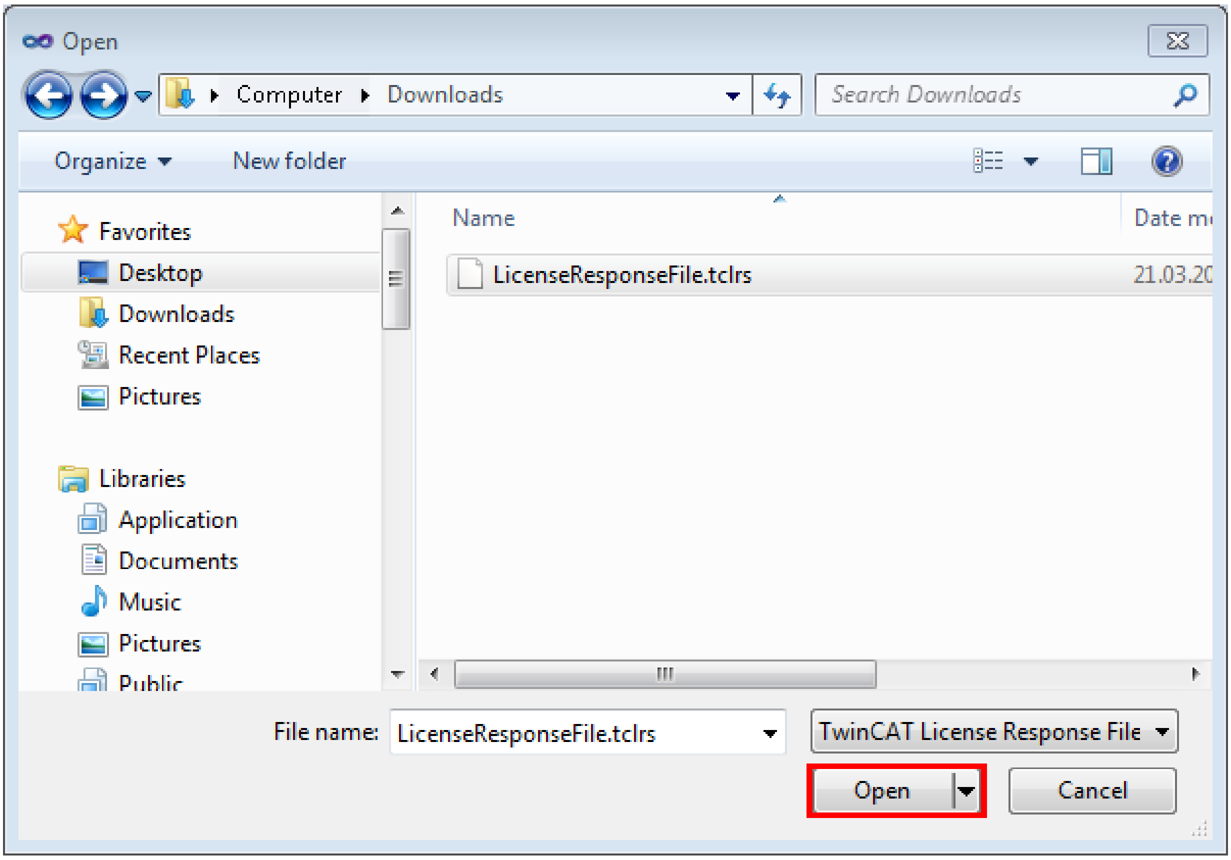
- Click **Yes** to send the License Request File. A prerequisite is that an email program is installed on your computer and that your computer is connected to the internet. When you click **Yes**, the system automatically generates a draft email containing the License Request File with all the necessary information.
- Click **No** if your computer does not have an email program installed on it or is not connected to the internet. Copy the License Request File onto a data storage device (e.g. a USB stick) and send the file from a computer with internet access and an email program to the Beckhoff license server (tclicense@beckhoff.com) by email.

10. Send the License Request File.

    ⇨ The License Request File is sent to the Beckhoff license server. After receiving the email, the server compares your license request with the specified order number and returns a License Response File by email. The Beckhoff license server returns the License Response File to the same email address from which the License Request File was sent. The License Response File differs from the License Request File only by a signature that documents the validity of the license file content. You can view the contents of the License Response File with an editor suitable for XML files (e.g. “XML Notepad”). The contents of the License Response File must not be changed, otherwise the license file becomes invalid.

11. Save the License Response File.

12. To import the license file and activate the license, click **License Response File...** in the **Order Information** tab.
13. Select the License Response File in your file directory and confirm the dialog.



File name: LicenseResponseFile.tclrs

TwinCAT License Response File

⇨ The License Response File is imported and the license it contains is activated. Existing demo licenses will be removed.

14. Restart the TwinCAT system.

⇨ The license becomes active when TwinCAT is restarted. The product can be used as a full version. During the TwinCAT restart the license file is automatically copied to the directory ...*\TwinCAT\3.1\Target\License* on the respective target system.

3.4 Installation Windows CE

The following section describes how to install a TwinCAT 3 function (TFxxx) on a Beckhoff Embedded PC with Windows CE.

1. Download and install the setup file [▶ 16]
2. Transfer the CAB file to the Windows CE device [▶ 17]
3. Run the CAB file on the Windows CE device [▶ 17]

If an older TFxxx version is already installed on the Windows CE device, it can be updated:

- Software upgrade [▶ 17]

Download and install the setup file

The CAB installation file for Windows CE is part of the TFxxx setup. This is made available on the Beckhoff website www.beckhoff.com and automatically contains all versions for Windows XP, Windows 7 and Windows CE (x86 and ARM).

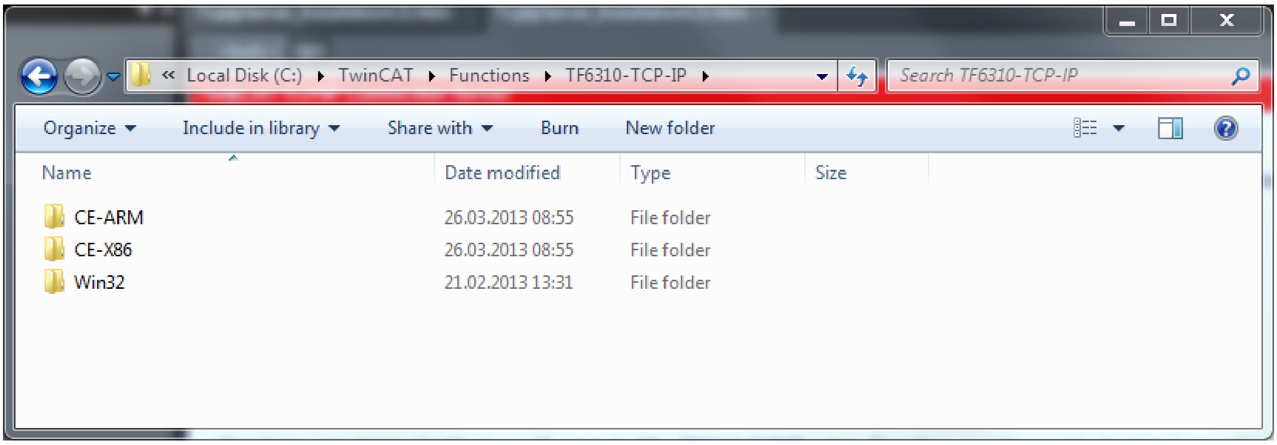
Download the TFxxx setup file and install the TwinCAT 3 function as described in the Installation [▶ 8] section.

After the installation, the installation folder contains three directories (one directory per hardware platform):

- **CE-ARM:** ARM-based Embedded PCs running Windows CE, e.g. CX8090, CX9020
- **CE-X86:** X86-based Embedded PCs running Windows CE, e.g. CX50xx, CX20x0
- **Win32:** Embedded PCs running Windows XP, Windows 7 or Windows Embedded Standard

The CE-ARM and CE-X86 directories contain the CAB files of the TwinCAT 3 function for Windows CE in relation to the respective hardware platform of the Windows CE device.

Example: "TF6310" installation folder



Name	Date modified	Type	Size
CE-ARM	26.03.2013 08:55	File folder	
CE-X86	26.03.2013 08:55	File folder	
Win32	21.02.2013 13:31	File folder	

Transfer the CAB file to the Windows CE device

Transfer the corresponding CAB file to the Windows CE device.

There are various options for transferring the executable file:

- via network shares
- via the integrated FTP server
- via ActiveSync
- via CF/SD cards

Further information can be found in the Beckhoff Information System in the "Operating Systems" documentation (Embedded PC > Operating Systems > CE).

Run the CAB file on the Windows CE device

After transferring the CAB file to the Windows CE device, double-click the file there. Confirm the installation dialog with **OK**. Then restart the Windows CE device.

After restarting the device, the files of the TwinCAT 3 function (TFxxxx) are automatically loaded in the background and are then available.

The software is installed in the following directory on the Windows CE device:
\Hard Disk\TwinCAT\Functions\TFxxxx

Software upgrade

If an older version of the TwinCAT 3 function is already installed on the Windows CE device, carry out the following steps on the Windows CE device to upgrade to a new version:

1. Open the CE Explorer by clicking **Start > Run** and entering "Explorer".
2. Navigate to *\Hard Disk\TwinCAT\Functions\TFxxx\xxxx*.
3. Rename the file *Tc*.exe* to *Tc*.old*.
4. Restart the Windows CE device.
5. Transfer the new CAB file to the Windows CE device.

6. Run the CAB file on the Windows CE device and install the new version.
7. Delete the file *Tc*.old*.
8. Restart the Windows CE device.

⇨ The new version is active after the restart.

4 Technical introduction

4.1 Basic concept

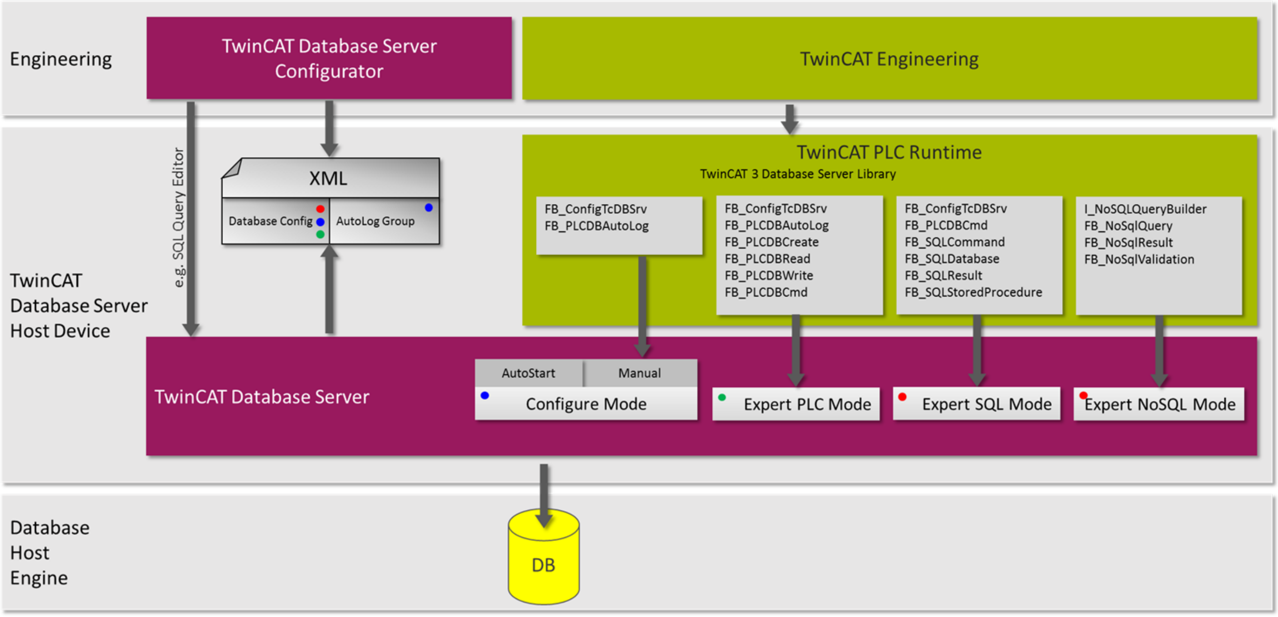
The TwinCAT Database Server is designed to enable a database connection to the controller for all TwinCAT users, and as conveniently as possible. Notwithstanding the required simplicity, the full flexibility is to be retained, which is why the TwinCAT Database Server offers three basic categories:

- **Configure mode:pure configuration solution**
  Database connections for simple applications based on graphical configurations without code implementation.
- **PLC Expert mode: programming solution for conventional PLC programmers**
  Database connection for simple or complex applications based on PLC function blocks, in which the database commands are largely generated automatically by the Database Server.
- **SQL Expert mode: programming solution for conventional PLC programmers and database experts**
  Database connection for simple or complex applications based on PLC function blocks and C++ interfaces, in which the database commands are assembled automatically during program execution. For maximum flexibility.
- **NoSQL Expert mode: programming solution for PLC programmers and NoSQL database experts**
  Database connection for simple to complex applications with PLC function blocks, in which NoSQL commands can be created and sent within the program sequence.

Naturally, all three categories can be combined within an application.

The TwinCAT Database Server can be set up via a graphical configurator. The configuration is written to an XML file, which can then be downloaded to the target system.
On non-Windows CE devices the configuration file is in folder *C:\TwinCAT\3.1\Boot*, on Windows CE devices in folder *\Hard Disk\TwinCAT\ 3.1\Boot*.
Read and write access are available for the different database systems. The supported databases are described in section "Databases [▶ 118]".

The TwinCAT Database Server service is started together with the TwinCAT system on the respective control computer. It is the link between the PLC and the database.



Configure Mode

In Configure mode, the bulk of the work is done in the configurator. The configuration has to be set up for the required database and for the AutoLog group. The target browser can be used for configuring the AutoLog group, for online access to a target system, and for selecting the variables to be communicated. If the **AutoStart** option is used, the communication with the configured database is established directly when TwinCAT system starts up. If the **Manual** option is selected, the communication has to be enabled via the function block FB_PLCDBAutoLog [▶ 149] or for AutoLog view.

PLC Expert mode

In PLC Expert mode only the database configuration is set in the configurator. Further functionalities are implemented in the PLC code of the application. With the function block FB_PLCDBCreate [▶ 161] it is possible to dispense with the configurator and even configure the database itself from the PLC. Function blocks for reading and writing are available, if required. The function block FB_PLCDBCmd [▶ 173] forms the transition between PLC Expert mode and SQL Expert mode. Here, table structures can easily be mapped as PLC structures, and an SQL command with placeholders for the current structure values can be transferred to the TwinCAT Database Server. The TwinCAT Database Server then inserts all values automatically and sends the command to the database.

SQL Expert Mode

In SQL Expert mode users can assemble the SQL commands for Insert, Select or Update, for example, in the PLC and send them to the database via the TwinCAT Database Server. This is a very flexible and powerful option. Stored Procedures [▶ 191] - in database - can also be called from the PLC.

Logging of structures

Note the corresponding byte alignment when logging structures.

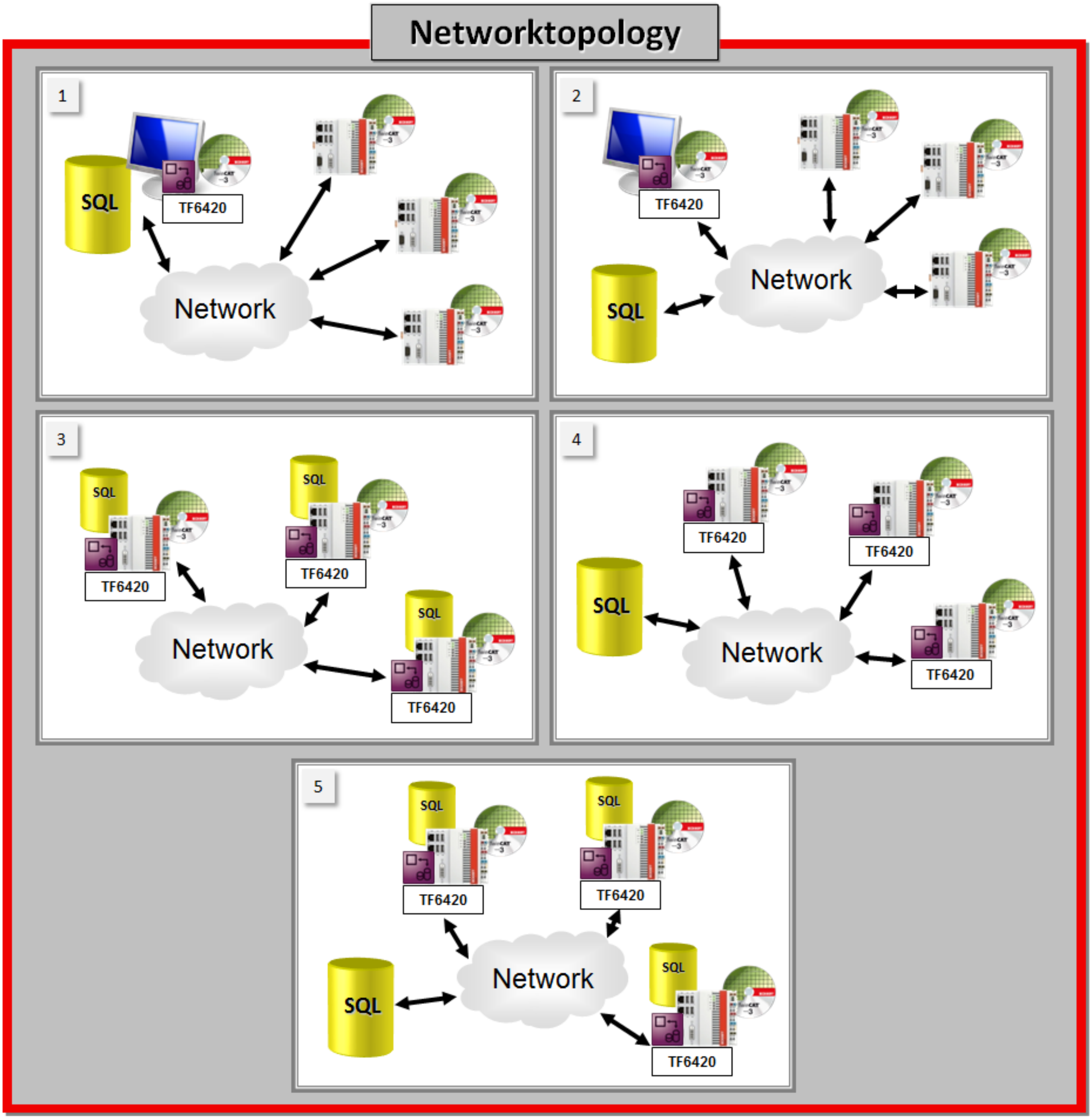
NoSql Expert Mode

In NoSQL Expert mode, the user can compile NoSQL queries such as Insert or Find and many other database-specific queries and send them to the database via the TwinCAT Database Server. New and more flexible data schemas, such as hierarchical structures and arrays, are supported.

4.2 Areas of application and network topologies

The TwinCAT Database Server can be used in any control application: reading recipe data in production machines, labelling products with production data, condition monitoring or machine control, logging of wind turbine operating states, or building services. The Database Server can be integrated in the existing network architecture.

The network topology is mostly influenced by the database type, the local conditions and the area of application. The following illustration shows various network topologies in which the TwinCAT Database Server can be used.



1. TwinCAT and the TwinCAT Database Server reside on the same computer, together with the database. The Database Server can act as gateway for many controllers via ADS. The performance must be taken into account.
2. TwinCAT and the TwinCAT Database Server reside on the same computer, while the database is on an external device. Here, too, the Database Server can act as gateway for many controllers via ADS. The performance must be taken into account.
3. The TwinCAT Database Server resides locally on each control device that has a database installed. Not all databases are suitable for this kind of application.
4. This is the most common use case. The TwinCAT Database Server is installed on each control device, and the database resides on an external server in the network.
5. Combination of case 3 and case 4. A main database resides on a server in the network, and the controllers in the field each have a local database, which kicks in when a disconnection is encountered, for example, and stores the data locally in the first instance. The Database Server is installed on each control device.

Remote access by the TwinCAT Database Server to a database

For remote access by the TwinCAT Database Server to a database, various aspects have to be taken into account on the database side:

- Is remote access generally permitted?
- How many simultaneous connections are permitted? (In case the TwinCAT Database Server needs to open several connections)
- Does the user who wishes to log onto the database with the Database Server have sufficient rights?
- Is the firewall of the remote system configured appropriately?

More detailed information about the configuration of your database server can be found in the corresponding database documentation [▶ 118].

4.3 Compatibility

The TwinCAT Database Server is a tried and tested TwinCAT product that has been around for many years. The demands on the product are constantly increasing. New developments in the TwinCAT Database Server are intended to meet these increased requirements.

The TwinCAT database connections have previously been available in versions 3.**0**.x and 3.**1**.x. The new functionalities, such as NoSQL support and the update of the PLC function blocks based on the new EventLogger interface are available from version number 3.**2**.x. As before, the Database Server consists of the components configurator, ADS server and PLC library. Version 3.0.x includes the PLC library Tc2_Database.compiled library. The PLC library in versions 3.1.x and 3.2.x is called Tc3_Database.compiled-library.

Overview of released Database Server versions

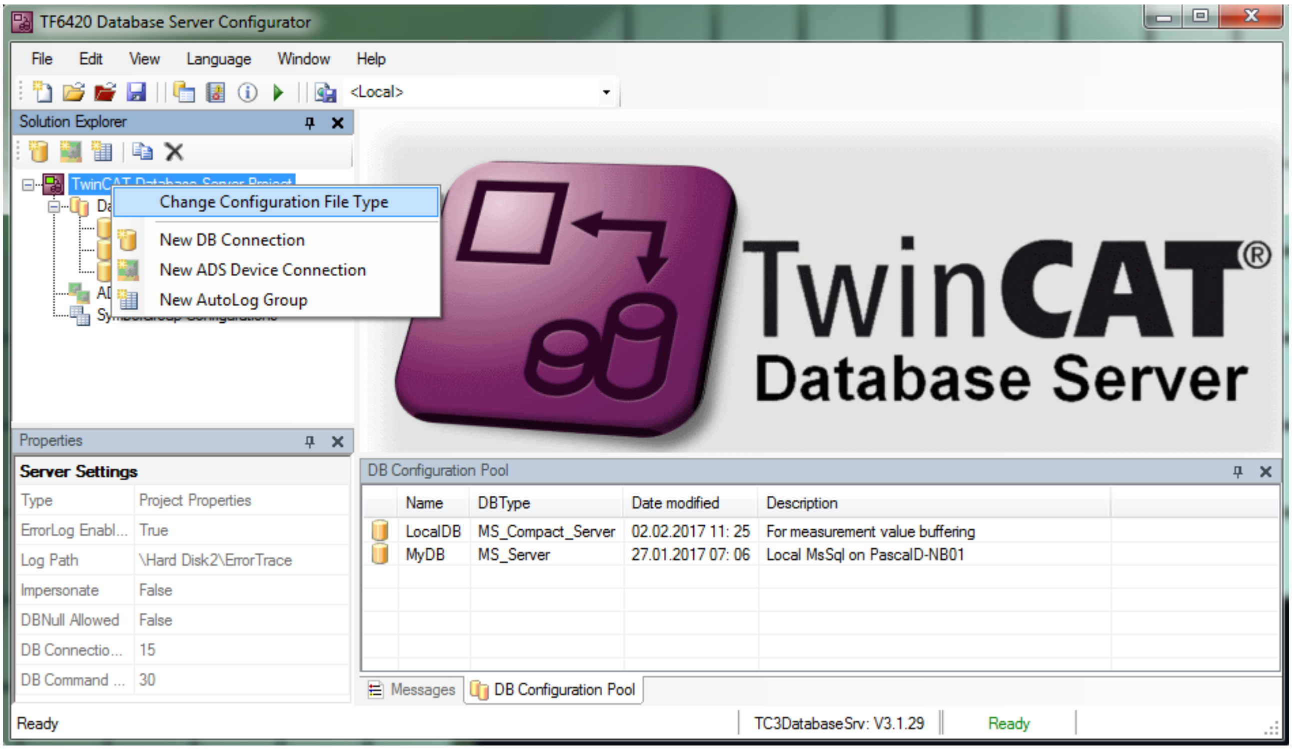
Database Server 3.0.x	3.0.23	3.0.26	3.0.27	3.0.28				
Database Server 3.1.x					3.1.29	3.1.30	3.1.31	
Database Server 3.2.x								3.2.32

Notes on the transition from 3.0.x to 3.1.x

In addition to new and higher-performance functions, a key aspect was compatibility between versions 3.0.x and 3.1.x. For example, old PLC code, in which the Tc2_Database.compiled library is used, can also be used with the new 3.1.x version ADS server. The old Tc2_Database.compiled library continues to be installed in version 3.1.x during setup. The XML files created by the configurator for the server differ between versions 3.0.x and 3.1.x. It is possible to read old configuration files with the new configurator (standalone) and even to convert them to the new format, if required.

Backup of the old XML configuration

During an update from the TwinCAT Database Server 3.0.x to the new 3.1.x version, the old XML configuration is saved. It is renamed to "CurrentConfigDataBase_**OLD**.xml" and remains in the TwinCAT boot directory.

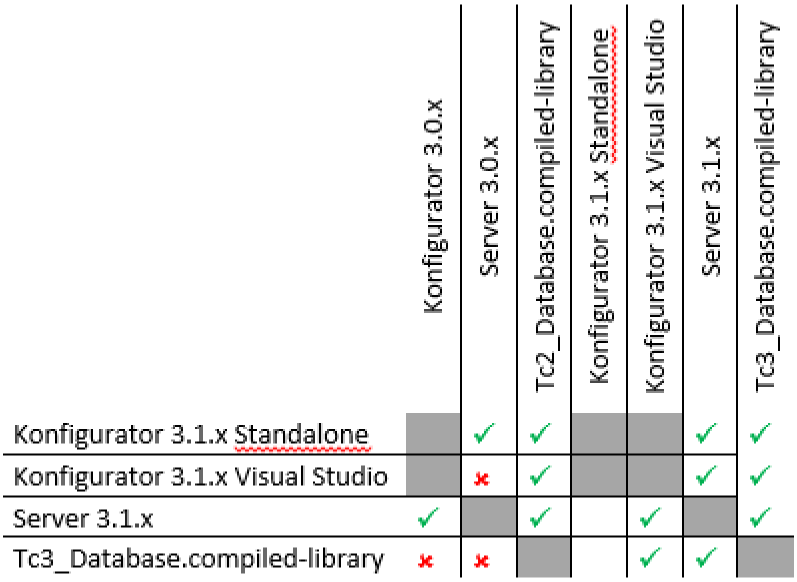


Server Settings

Type	Project Properties
ErrorLog Enabl...	True
Log Path	\Hard Disk2\ErrorTrace
Impersonate	False
DBNull Allowed	False
DB Connectio...	15
DB Command ...	30

Name	DBType	Date modified	Description
LocalDB	MS_Compact_Server	02.02.2017 11: 25	For measurement value buffering
MyDB	MS_Server	27.01.2017 07: 06	Local MsSql on PascalD-NB01

Notwithstanding the general compatibility referred to above, an old configurator and an old ADS server (version 3.0.x) cannot be used with the new Tc3_Database.compiled library. The diagram below provides a compatibility overview.



	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	✗	✗			✓	✓	

Notes on the transition from 3.1.x to 3.2.x

The file formats for the configurations are unchanged. The ADS server was merely extended with new functionalities. All other functions are still available. In version 3.2.x the old Tc2_Database.compiled-library is installed in parallel with the Tc3_Database.compiled-library during setup. The notes for the transition from 3.0.x to 3.1.x apply.

In the Tc3_Database.compiled-library, all previous function blocks have been updated from version 3.2.x onwards. The update refers to the I_TcMessage EventLogger interface. To ensure that older applications continue to function, "Evt" is appended to the names of new function blocks. All old function blocks are still contained in the library, but are now in the Obsolete folder and are marked accordingly by the compiler.

Sample:

Note for version 3.1.x: FB_SQLCommand

Note for version 3.2.x: FB_SQLCommandEvt

We recommend using the function blocks with the ending "Evt" for new projects. It should be noted that the EventLogger itself is only available from TC 3.1 Build 4022.20, and therefore the function blocks can only be used from 4022.20.

5 Configuration

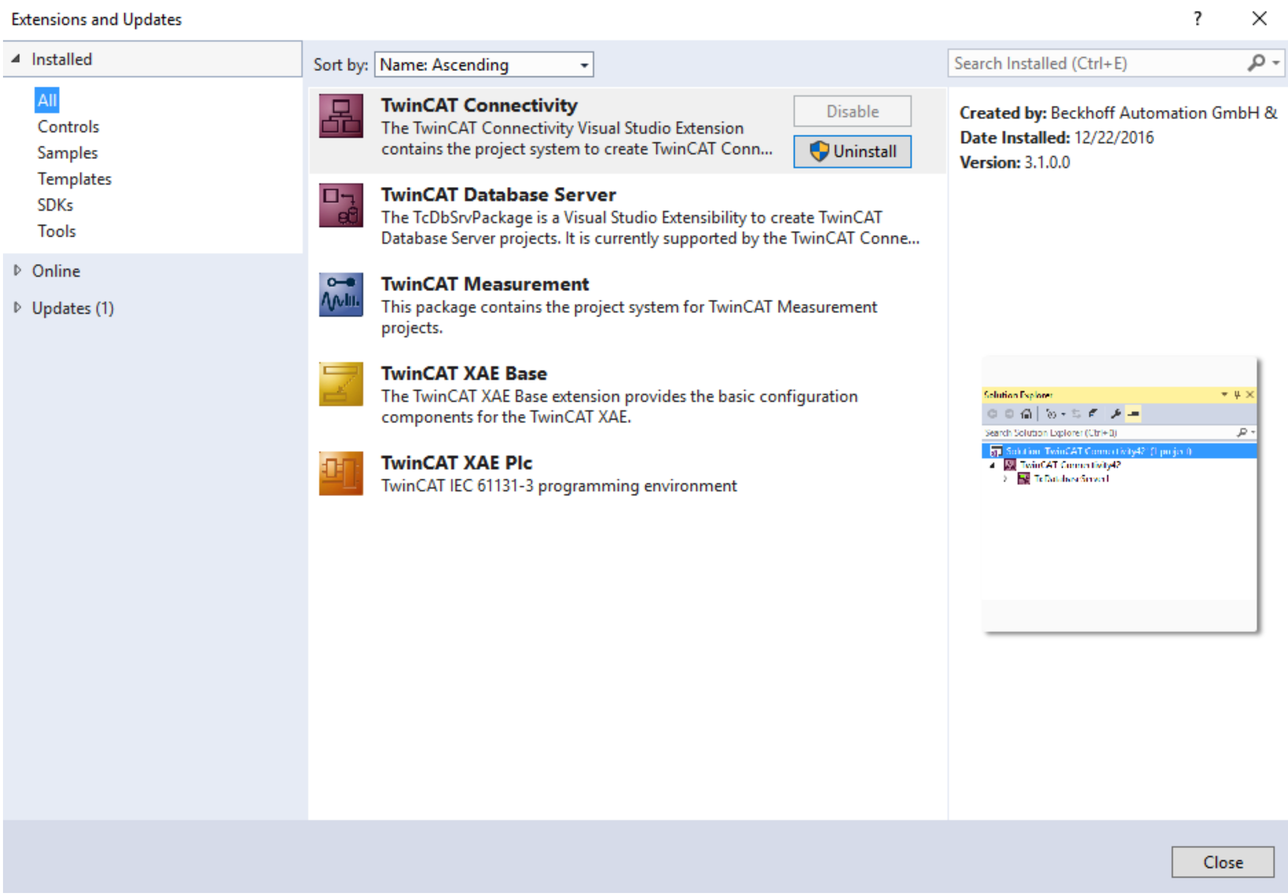
5.1 Configurator

The TwinCAT Database Server is set and controlled via the configurator. The tool also offers a range of development facilities for speeding up the development of the application in the PLC.

The configurator is integrated in Visual Studio, in order to make development as user-friendly as possible. TwinCAT projects and TwinCAT Database Server projects can be placed in a common solution. Alternatively, customers can continue to use the standalone configurator, independent of Visual Studio.

5.1.1 Integration in Visual Studio

The TwinCAT Database Server is integrated in Visual Studio 2013 and Visual Studio 2015. This integration is achieved with the aid of two extensions. The two required extensions are added in the Visual Studio extension window during the installation, which contain the functionalities of the project system, among other features.



Sort by: Name: Ascending

TwinCAT Connectivity
The TwinCAT Connectivity Visual Studio Extension contains the project system to create TwinCAT Conn...

TwinCAT Database Server
The TcDbSrvPackage is a Visual Studio Extensibility to create TwinCAT Database Server projects. It is currently supported by the TwinCAT Conne...

TwinCAT Measurement
This package contains the project system for TwinCAT Measurement projects.

TwinCAT XAE Base
The TwinCAT XAE Base extension provides the basic configuration components for the TwinCAT XAE.

TwinCAT XAE Plc
TwinCAT IEC 61131-3 programming environment

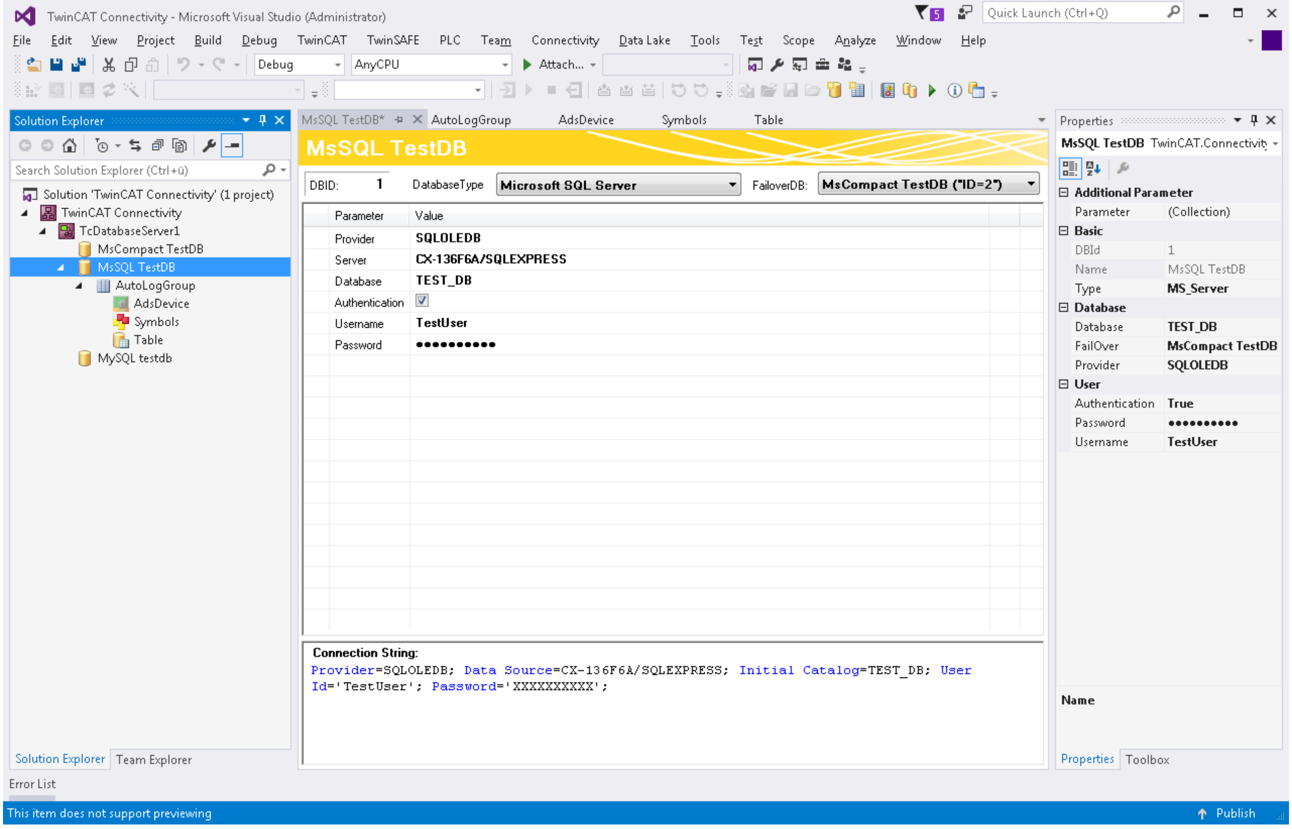
Created by: Beckhoff Automation GmbH &
Date Installed: 12/22/2016
Version: 3.1.0.0

5.1.1.1 General

This chapter describes the various functions and components of the Visual Studio integration of the TwinCAT Database Server.

5.1.1.1.1 User interface components

The TwinCAT Database Server is integrated in Visual Studio 2013 and Visual Studio 2015. The TwinCAT Connectivity extension for Visual Studio offers a new project system. This can be used for creating a file-based TwinCAT Database Server project, for example. Typical components such as the Properties window, the Solution Explorer and the error output are supported. In addition, various editors for editing the configuration files are provided.


TwinCAT Connectivity - Microsoft Visual Studio (Administrator)
Quick Launch (Ctrl+Q)
File Edit View Project Build Debug TwinCAT TwinSAFE PLC Team Connectivity Data Lake Tools Test Scope Analyze Window Help
Debug
AnyCPU
Attach...
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connectivity' (1 project)
TwinCAT Connectivity
TcDatabaseServer1
MsCompact TestDB
MsSQL TestDB
AutoLogGroup
AdsDevice
Symbols
Table
MySQL testdb
Solution Explorer
Team Explorer
MsSQL TestDB*
AutoLogGroup
AdsDevice
Symbols
Table
MsSQL TestDB
Properties
Toolbox
Error List
This item does not support previewing
Publish

DBID: 1 DatabaseType: Microsoft SQL Server FailoverDB: MsCompact TestDB ("ID=2")

Parameter	Value
Provider	SQLOLEDB
Server	CX-136F6A/SQLEXPRESS
Database	TEST_DB
Authentication	☑
Username	TestUser
Password	••••••••••

Connection String:

```
Provider=SQLOLEDB; Data Source=CX-136F6A/SQLEXPRESS; Initial Catalog=TEST_DB; User
Id='TestUser'; Password='XXXXXXXXXX';
```

Properties: MsSQL TestDB TwinCAT.Connectivity

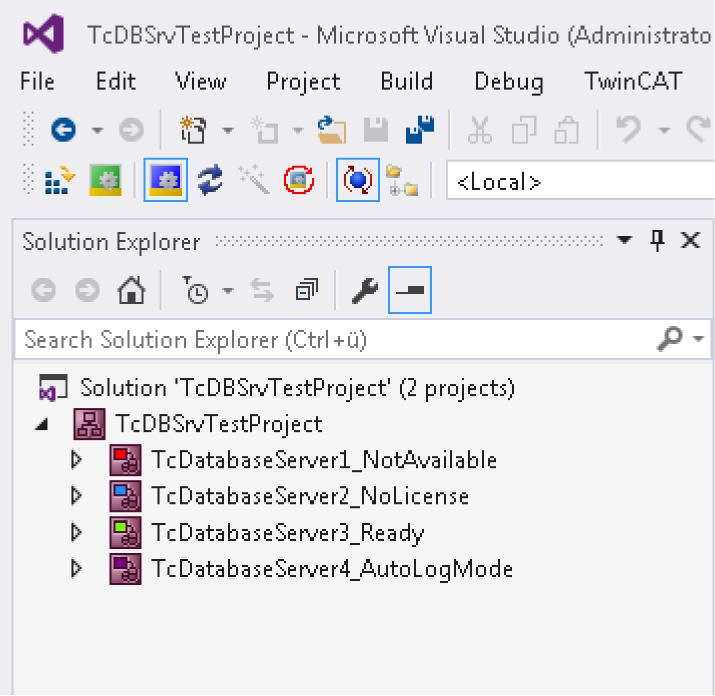
Additional Parameter	
Parameter	(Collection)
Basic	
DBId	1
Name	MsSQL TestDB
Type	MS_Server
Database	
Database	TEST_DB
FailOver	MsCompact TestDB
Provider	SQLOLEDB
User	
Authentication	True
Password	••••••••••
Username	TestUser

Name

Any number of TwinCAT Database Server projects can be integrated in a TwinCAT connectivity project.

The project icon indicates the state of the set target system:

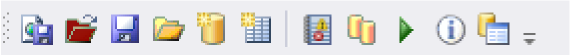
- Red: The TwinCAT Database Server cannot be reached.
- Blue: The TwinCAT Database Server has no valid license. (See Licensing [▶ 11])
- Green: The TwinCAT Database Server can be reached and is ready for use.
- Violet: The TwinCAT Database Server is in AutoLog mode. Data are exchanged between PLC and database.


TcDBSrvTestProject - Microsoft Visual Studio (Administrato
File Edit View Project Build Debug TwinCAT
<Local>
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TcDBSrvTestProject' (2 projects)
TcDBSrvTestProject
TcDatabaseServer1_NotAvailable
TcDatabaseServer2_NoLicense
TcDatabaseServer3_Ready
TcDatabaseServer4_AutoLogMode

The TwinCAT Database projects map a file-based project system. The individual configuration documents are managed in the Solution Explorer. Any modifications that are pending in the editors are identified with * in the documents. If changes are made without opening a document (through the Properties window), the changes are nevertheless registered. Further information on the project system can be found in section "TwinCAT Database Server project [▸ 29]".

Toolbar and commands

The toolbar has the following elements:

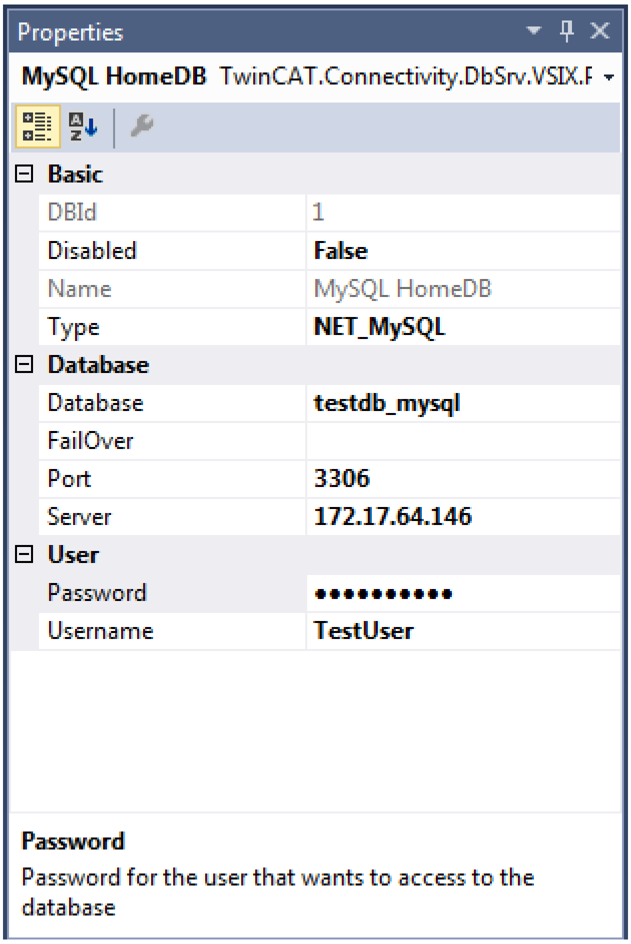


Toolstrip button	Description
	Activation of the configuration
📂	Read configuration of the target device
💾	Save configuration in an XML file
📂	Read configuration from an XML file
	Add new database configuration
	Add new AutoLog group
	Event display
	Database pool
▶	AutoLog Viewer
ⓘ	InformationLog View
	SQL Query Editor

Properties window

The settings for the different project documents can be configured via dedicated editors or via the Properties window. During this process the file content is modified, but not the metadata in the project file of the TwinCAT Connectivity project.

The individual properties are described in more detail in the lower part of the Properties window. Note that some lists can only be edited in the editor.



MySQL HomeDB TwinCAT.Connectivity.DbSrv.VSIX.F

Basic	
DBId	1
Disabled	False
Name	MySQL HomeDB
Type	NET_MySQL
Database	
Database	testdb_mysql
FailOver	
Port	3306
Server	172.17.64.146
User	
Password	●●●●●●●●●●
Username	TestUser

Password
Password for the user that wants to access to the database

Output and error list

Visual Studio features an integrated console output. The TwinCAT Database Server uses this feature to issue notifications, warnings or error messages. To this end the category "TwinCAT Database Server" can be selected in the output. It is possible that this category does not yet exist, if there was no previous message from the TwinCAT Database Server.

In addition, the Visual Studio error list is used for communicating the main information.

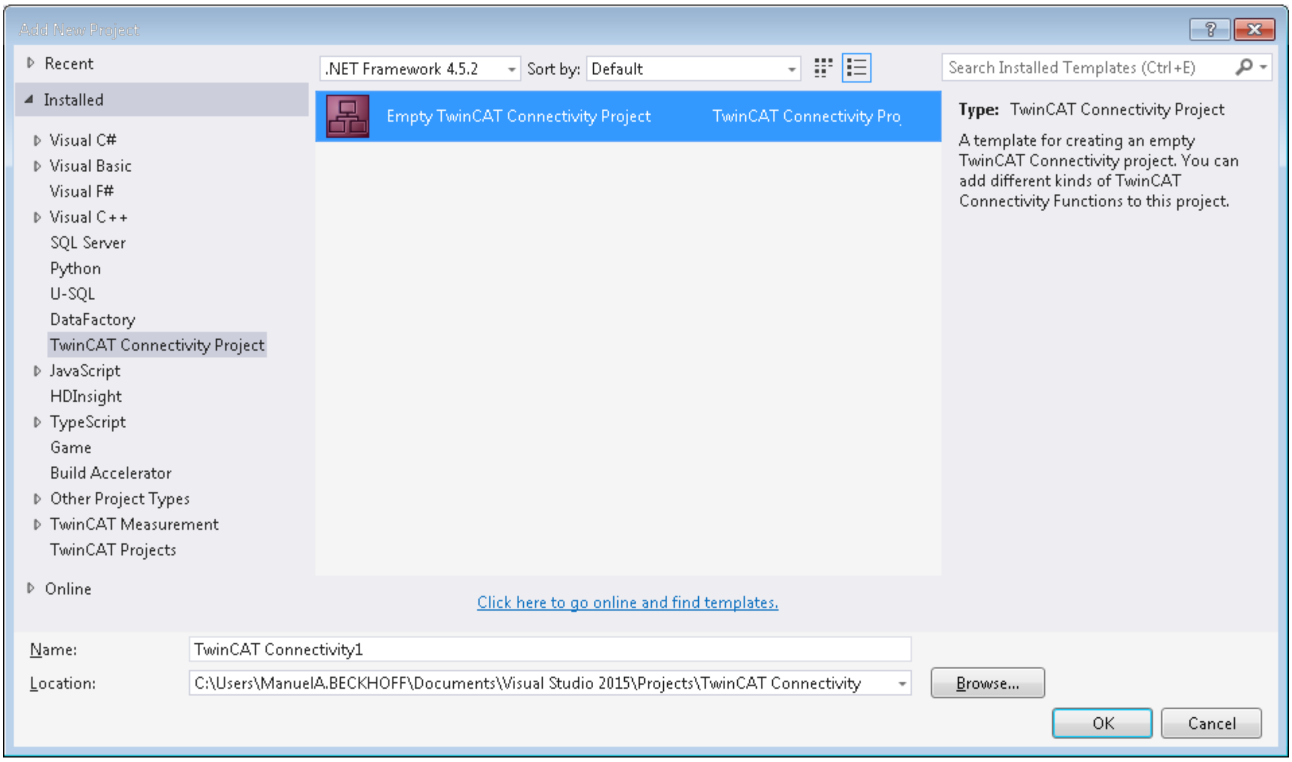
Both windows can be opened in Visual Studio via **View > Error list/Output**.

5.1.1.1.2 TwinCAT Database Server Project

Build Project

The TwinCAT Connectivity extension for Visual Studio provides a new project template. When a new project is created, the **TwinCAT Connectivity Project** category appears as an option.

To create a new TwinCAT Connectivity project, select **Empty TwinCAT Connectivity Project**, specify the project name and the storage location and click **OK** to add it to the solution. In this way, TwinCAT Connectivity projects or TwinCAT Database Server projects can conveniently be created in parallel with TwinCAT or other Visual Studio projects.



.NET Framework 4.5.2

Sort by: Default

Type: TwinCAT Connectivity Project

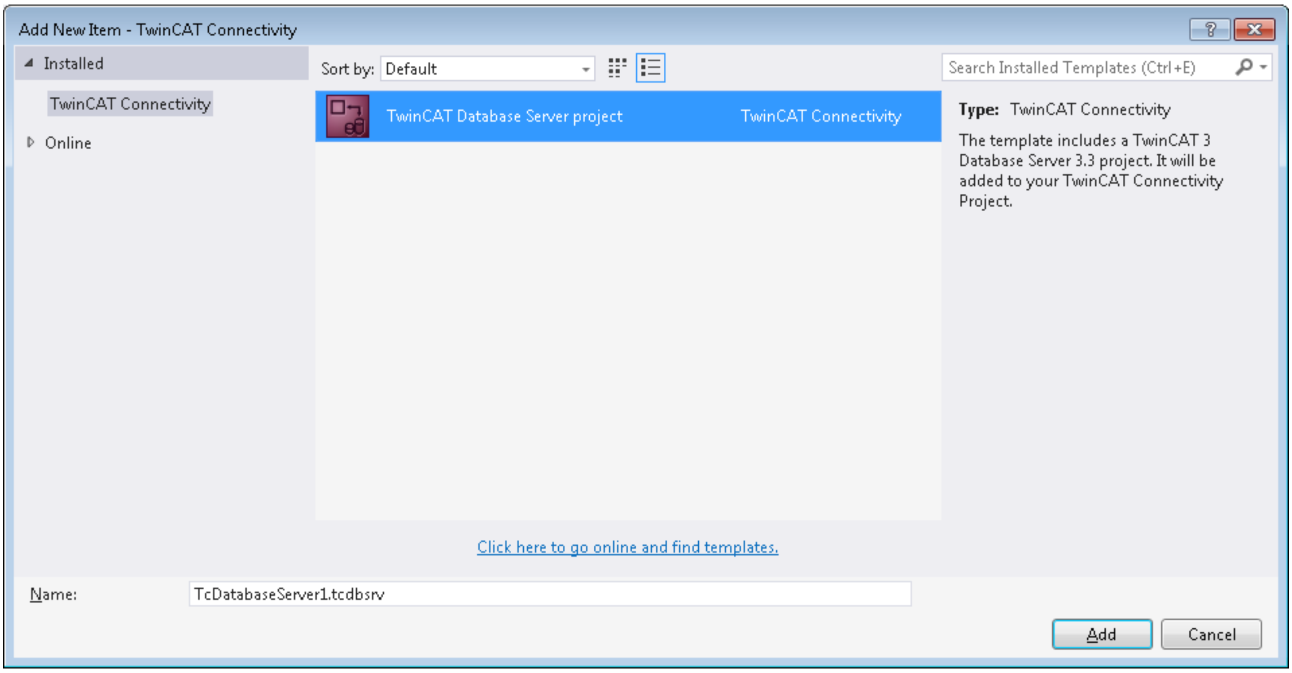
A template for creating an empty TwinCAT Connectivity project. You can add different kinds of TwinCAT Connectivity Functions to this project.

Name: TwinCAT Connectivity1

Location: C:\Users\ManuelA.BECKHOFF\Documents\Visual Studio 2015\Projects\TwinCAT Connectivity

A new project node appears in the solution. Below the Connectivity project node you can add subprojects for the supported connectivity functions.

Use **Add** to add a new TwinCAT Database Server project to the TwinCAT Connectivity project. The TwinCAT Database Server project can be found in the list of existing Item Templates.


Add New Item - TwinCAT Connectivity
Installed
TwinCAT Connectivity
Online
Search Installed Templates (Ctrl+E)
TwinCAT Database Server project
TwinCAT Connectivity
Click here to go online and find templates.
Add
Cancel

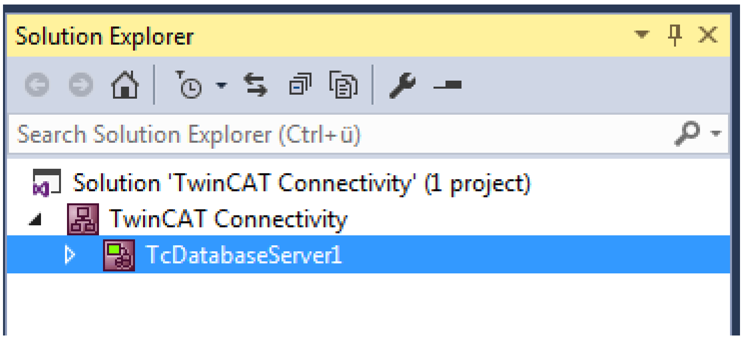
Sort by: Default

Type: TwinCAT Connectivity

The template includes a TwinCAT 3 Database Server 3.3 project. It will be added to your TwinCAT Connectivity Project.

Name: TcDatabaseServer1.tcdbsrv

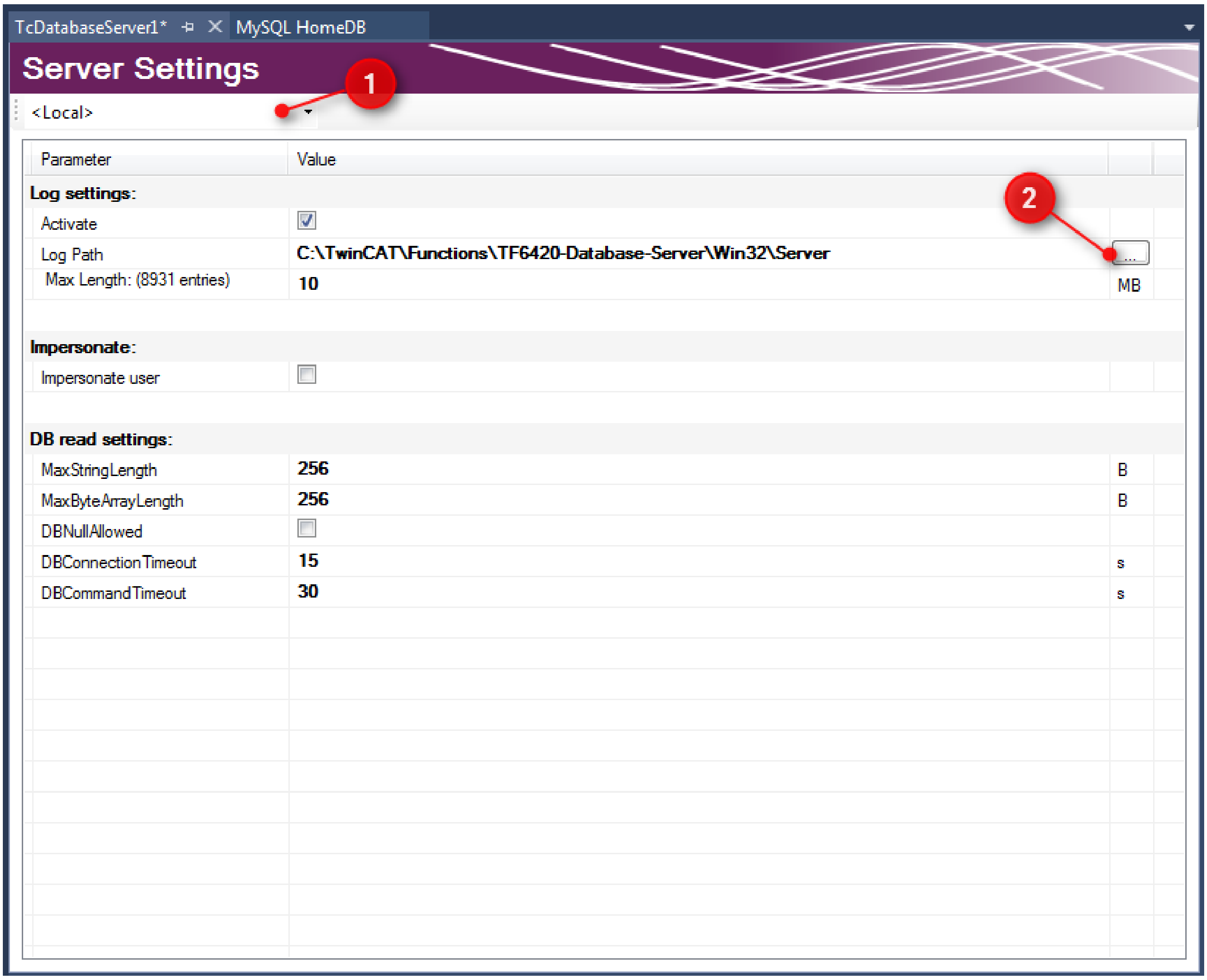
A new TwinCAT Database Server project is created under the TwinCAT Connectivity node.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connectivity' (1 project)
TwinCAT Connectivity
TcDatabaseServer1

This is now used as the basis for the pending configuration of a TwinCAT Database Server. The document can be edited either via the Properties window or via an editor.

A Connectivity project can be associated with any number of TwinCAT Database Server projects or other projects, and it may therefore contain several configurations.

Editor for server settings



Parameter	Value	
Log settings:		
Activate	☑	
Log Path	C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server	
Max Length: (8931 entries)	10	MB
Impersonate:		
Impersonate user	□	
DB read settings:		
MaxStringLength	256	B
MaxByteArrayLength	256	B
DBNullAllowed	□	
DBConnectionTimeout	15	s
DBCommandTimeout	30	s

The **Server Settings** editor can be used to edit the settings for the TwinCAT Database Server. These are general settings relating to the corresponding server. In the drop-down menu (1) you can select the target system via the Ams NetID. To this end you have to create a route to the target system via TwinCAT. When a finished configuration is transferred, the settings are stored in the TwinCAT Database Server for this target system.

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the Information Log Viewer [▶ 53]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.

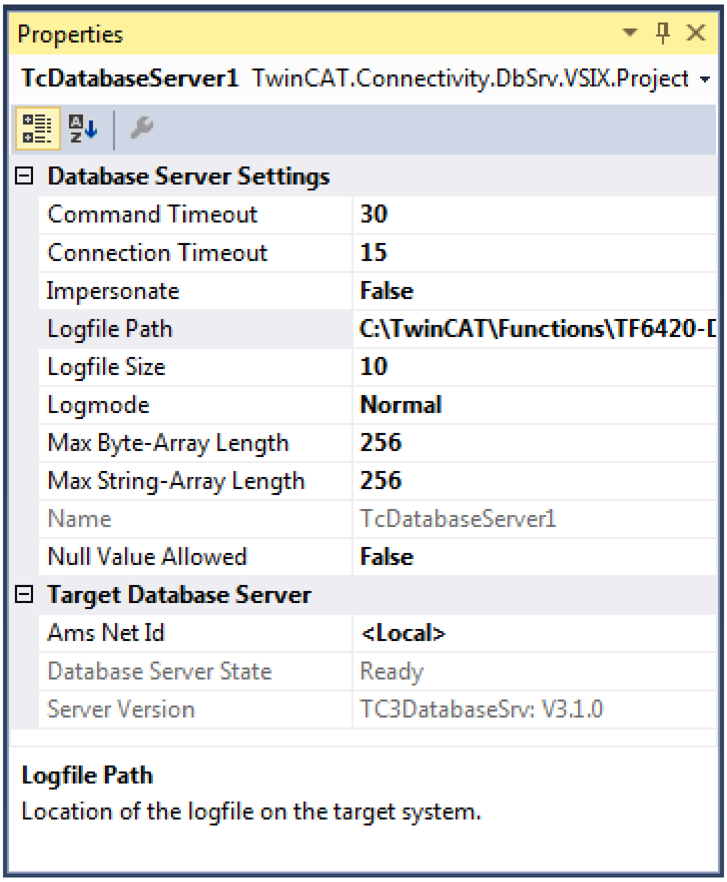
For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

Further configuration settings are available to control the read process from the database. These settings refer to the TwinCAT Database Server on the target system:

MaxStringLength	Maximum string length of the variables in the PLC
MaxByteArrayLength	Maximum byte array length of the variables in the PLC
DBNullAllowed	Indicates whether ZERO values are accepted in the TwinCAT Database Server.
DBConnectionTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection error while attempts are made to establish a connection.
DBCommandTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection fault when a command was sent. If large data quantities are involved, processing of a command may take quite some time, depending on the database and the infrastructure.

Server settings in the Properties window

The settings for the TwinCAT Database Server can be adjusted in the Editor window or in the Properties window of the Database Server. These properties also directly affect the configuration file.



Properties

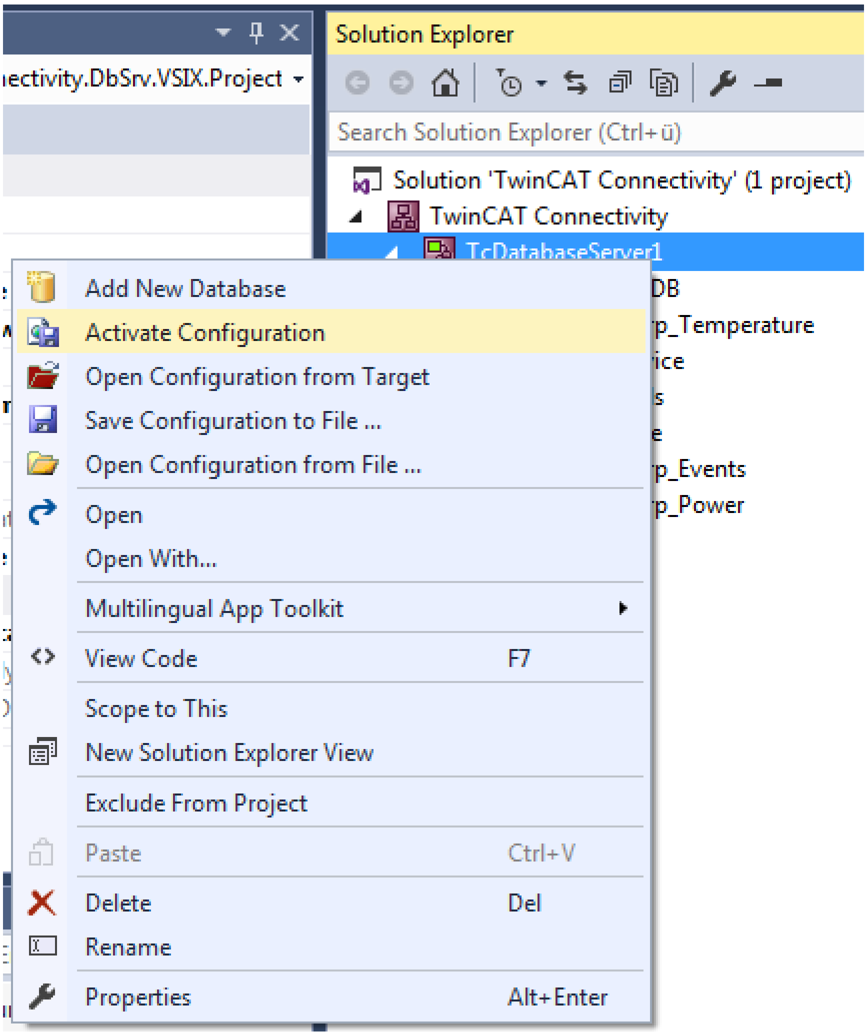
TcDatabaseServer1 TwinCAT.Connectivity.DbSrv.VSIX.Project

Database Server Settings	
Command Timeout	30
Connection Timeout	15
Impersonate	False
Logfile Path	C:\TwinCAT\Functions\TF6420-[
Logfile Size	10
Logmode	Normal
Max Byte-Array Length	256
Max String-Array Length	256
Name	TcDatabaseServer1
Null Value Allowed	False
Target Database Server	
Ams Net Id	<Local>
Database Server State	Ready
Server Version	TC3DatabaseSrv: V3.1.0

Logfile Path
Location of the logfile on the target system.

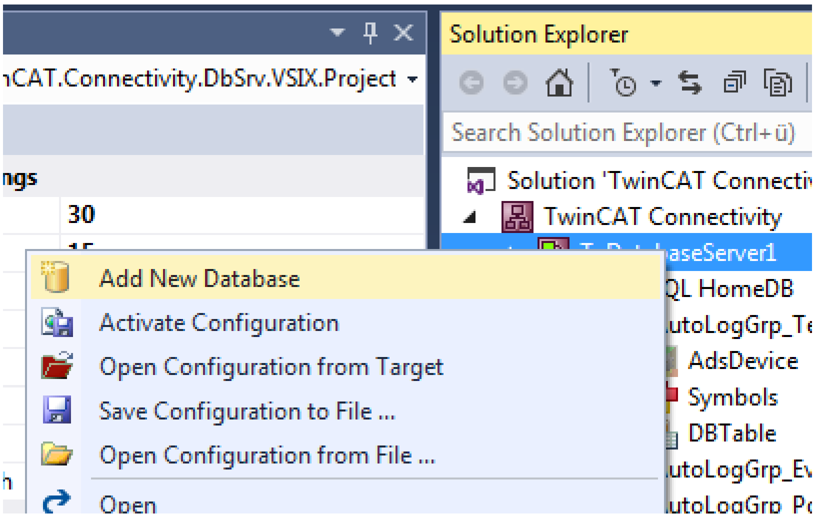
Activating a project

To activate a configured project on the TwinCAT Database Server, use the command **Activate Configuration** in the context menu of the TwinCAT Database Server project.



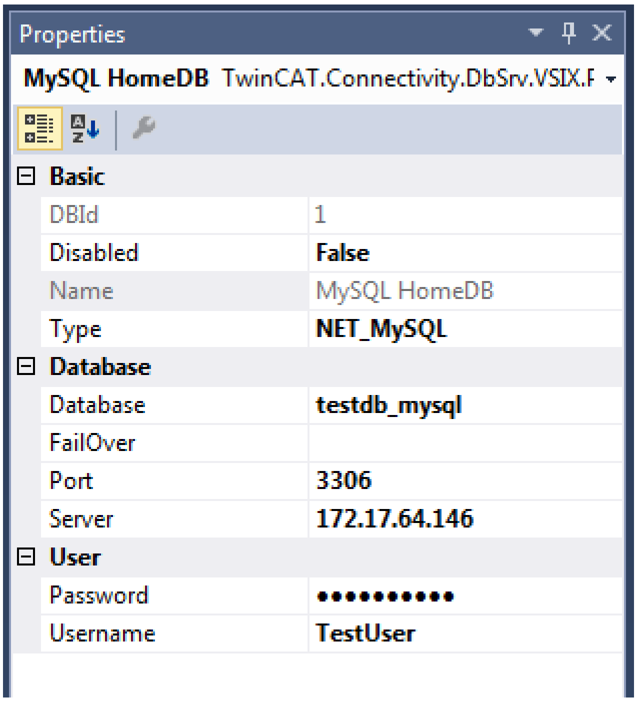
5.1.1.1.3 Configuring databases

A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connecti
TwinCAT Connectivity
Add New Database
Activate Configuration
Open Configuration from Target
Save Configuration to File ...
Open Configuration from File ...
Open
AdsDevice
Symbols
DBTable

A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

The new database configuration in the TwinCAT Database Server project can be edited via the Properties window or a special editor:

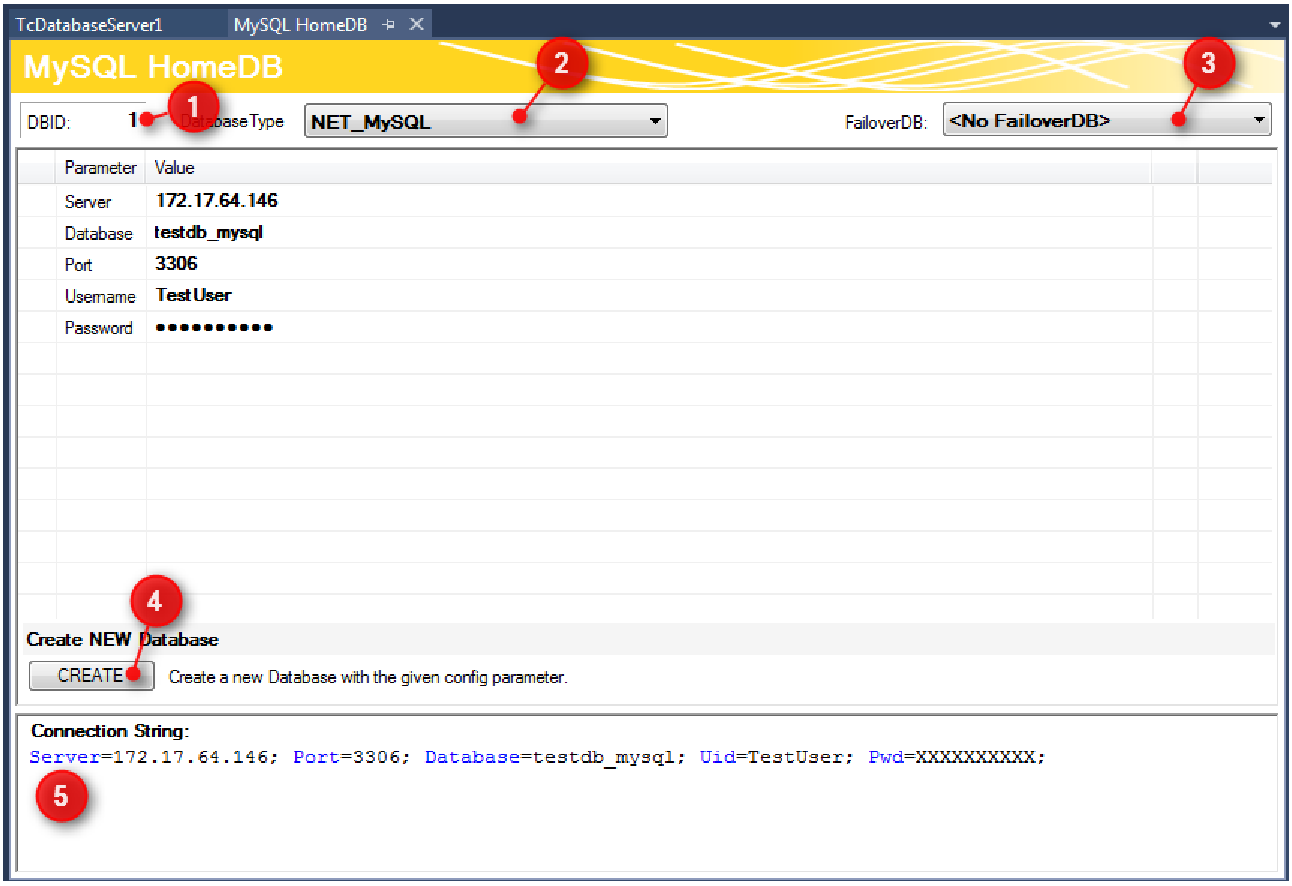


MySQL HomeDB TwinCAT.Connectivity.DbSrv.VSIX.F

Basic	
DBId	1
Disabled	False
Name	MySQL HomeDB
Type	NET_MySQL
Database	
Database	testdb_mysql
FailOver	
Port	3306
Server	172.17.64.146
User	
Password	••••••••••
Username	TestUser

The properties dynamically adapt to the selected database types, since the databases have different parameters. These settings relate to the file contents, not the file properties.

Editor for database configurations



MySQL HomeDB

DBID: 1 DatabaseType NET_MySQL FailoverDB: <No FailoverDB>

Parameter	Value
Server	172.17.64.146
Database	testdb_mysql
Port	3306
Username	TestUser
Password	••••••••••

Create NEW Database

CREATE Create a new Database with the given config parameter.

Connection String:

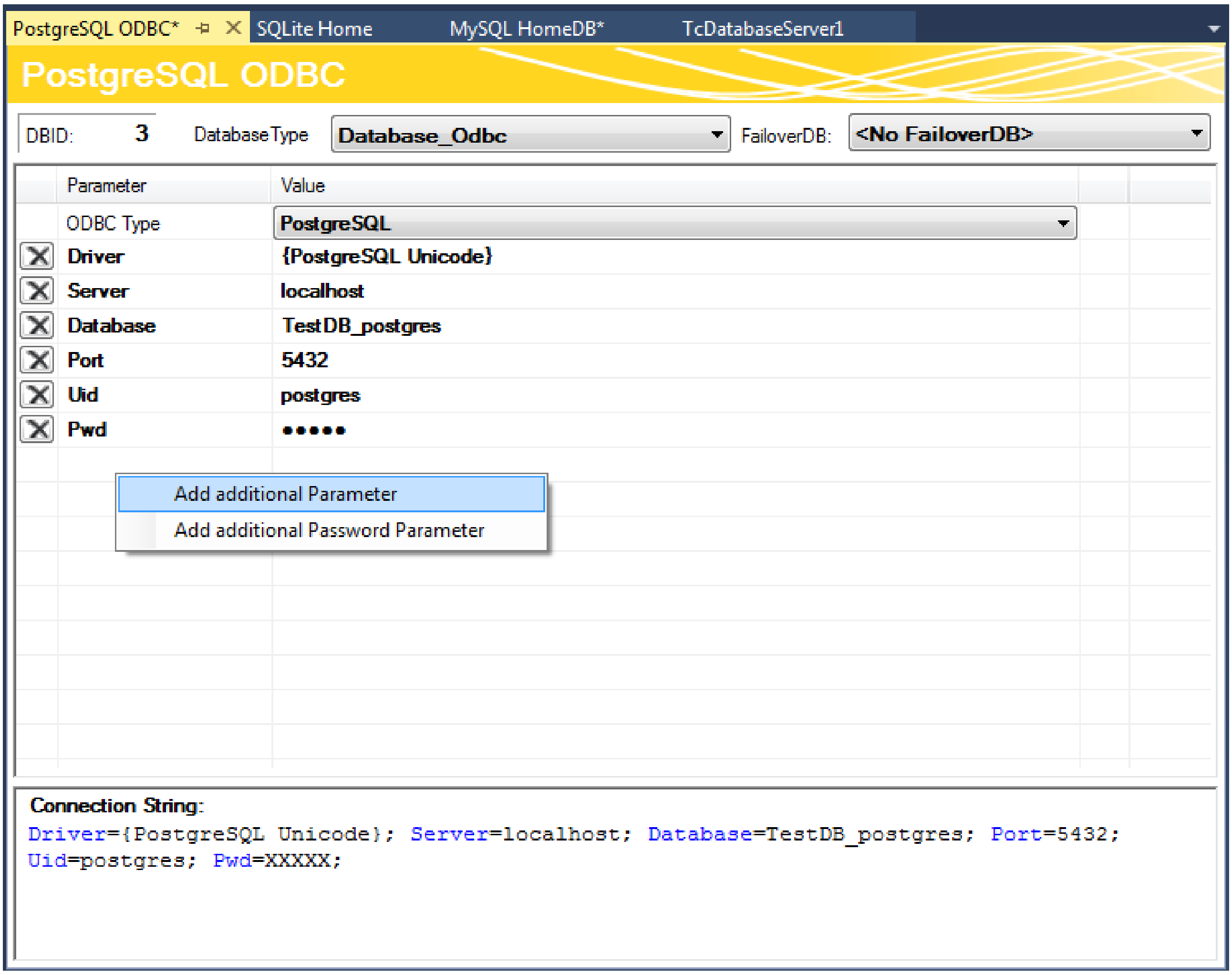
```
Server=172.17.64.146; Port=3306; Database=testdb_mysql; Uid=TestUser; Pwd=XXXXXXXXXX;
```

The database ID, which is required for some function blocks in the PLC, is shown in the upper part of the editor (1). The database types of the target database can be selected from the drop-down menu (2). Another option is the ODBC interface for a database, although this is not yet supported. Note that not all functions of the TwinCAT Database Server can be guaranteed, depending on the database.

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.

For each database [▶ 118] additional adjustable parameters are available. Depending on the database a connection string (5) is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

The **CREATE** (4) button can be used to create a new database. This function is only displayed if the respective database supports it.



DBID: 3 DatabaseType Database_Odbc FailoverDB: <No FailoverDB>

	Parameter	Value
	ODBC Type	PostgreSQL
☒	Driver	{PostgreSQL Unicode}
☒	Server	localhost
☒	Database	TestDB_postgres
☒	Port	5432
☒	Uid	postgres
☒	Pwd	●●●●●

Connection String:

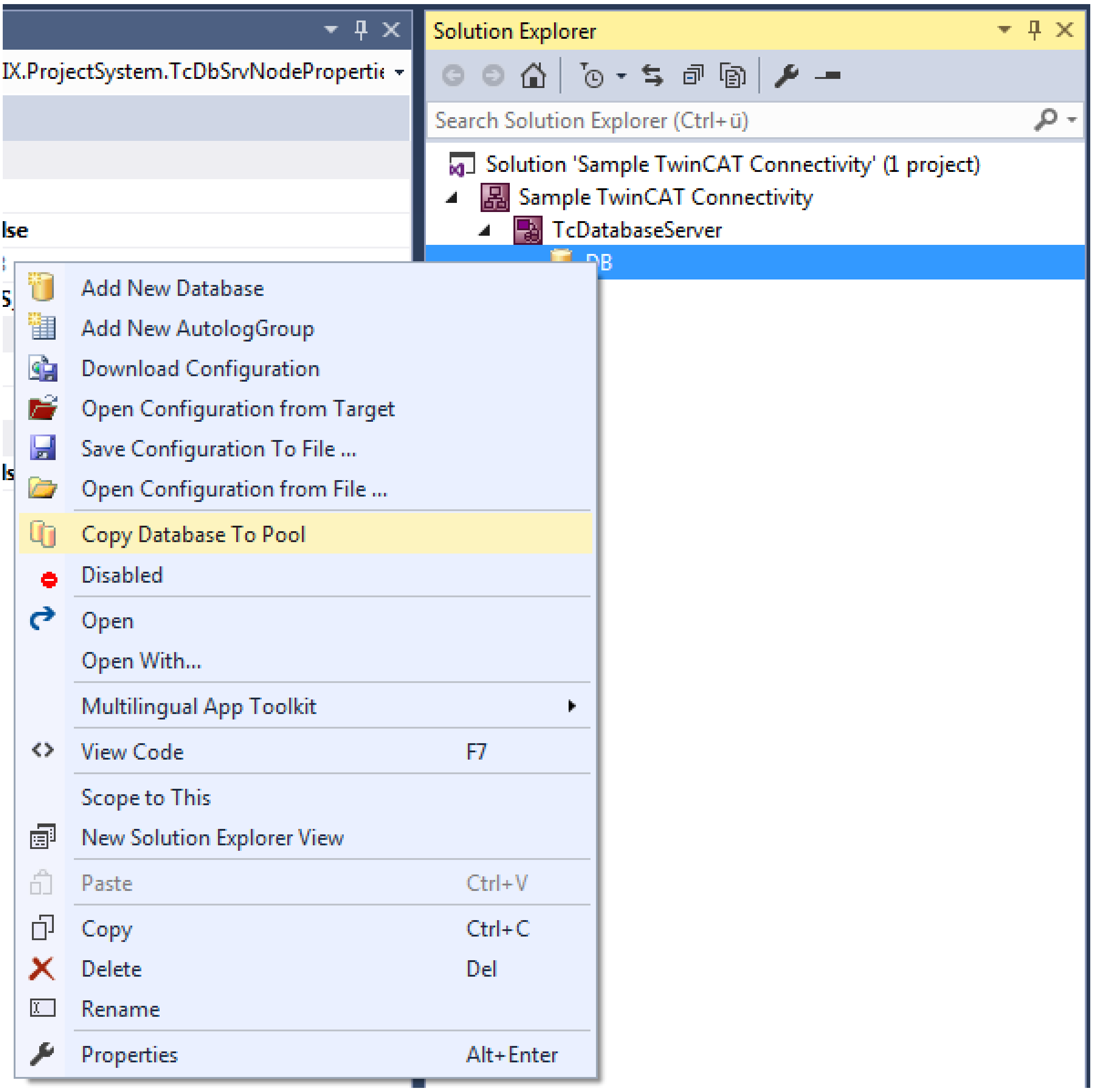
```
Driver={PostgreSQL Unicode}; Server=localhost; Database=TestDB_postgres; Port=5432;
Uid=postgres; Pwd=XXXXX;
```

Unknown databases can be configured via an ODBC interface. In the **ODBC Type** drop-down list select "Unknown Database" and add parameters via the commands in the context menu. They may contain passwords, which are stored in encrypted form. The required connection string can be assembled from these parameters. Note that only limited functions of the TwinCAT Database Server can be used. Only the explicit function blocks of the SQL Expert mode are supported.

The additional parameters can only be applied via the editor, not via the Properties window.

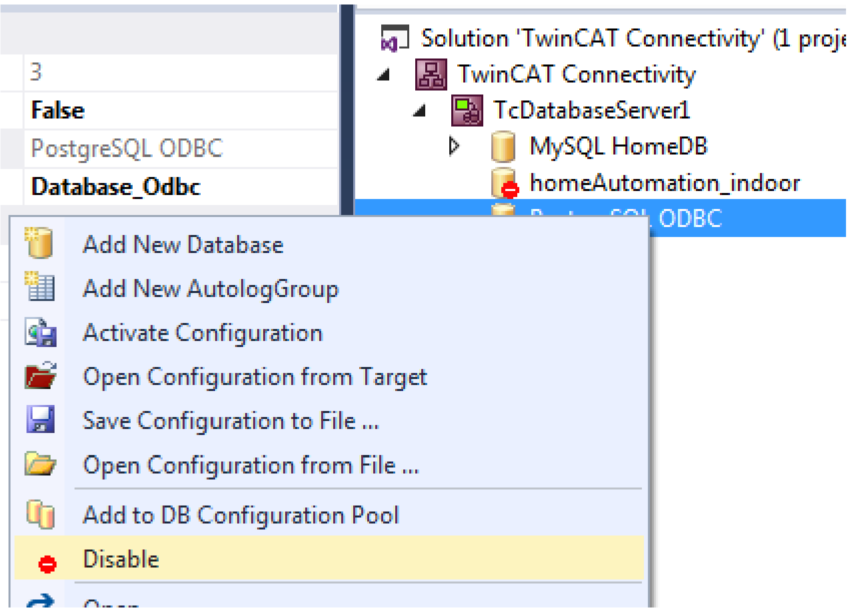
Copying a database configuration into the database pool

A corresponding command is available in the context menu for copying a database configuration into the database pool [▶ 53]. It is also possible to use drag & drop to move database configurations between the project and the database pool.


IX.ProjectSystem.TcDbSrvNodePropertie
lse
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'Sample TwinCAT Connectivity' (1 project)
Sample TwinCAT Connectivity
TcDatabaseServer
DB
Add New Database
Add New AutologGroup
Download Configuration
Open Configuration from Target
Save Configuration To File ...
Open Configuration from File ...
Copy Database To Pool
Disabled
Open
Open With...
Multilingual App Toolkit
View Code F7
Scope to This
New Solution Explorer View
Paste Ctrl+V
Copy Ctrl+C
Delete Del
Rename
Properties Alt+Enter

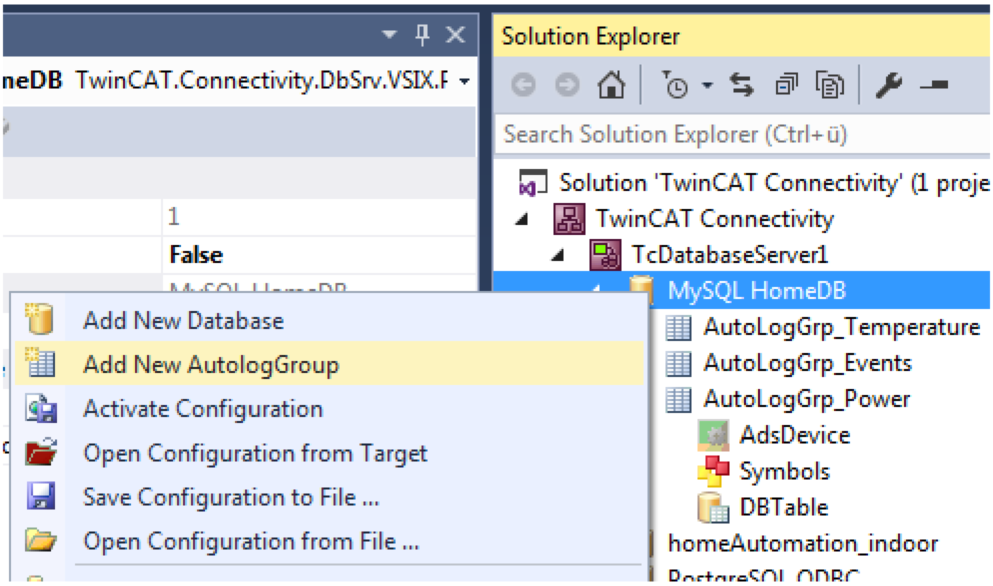
Deactivating database configurations

Individual database configurations can be disabled in the project. These are then marked in red and ignored when the project is activated.

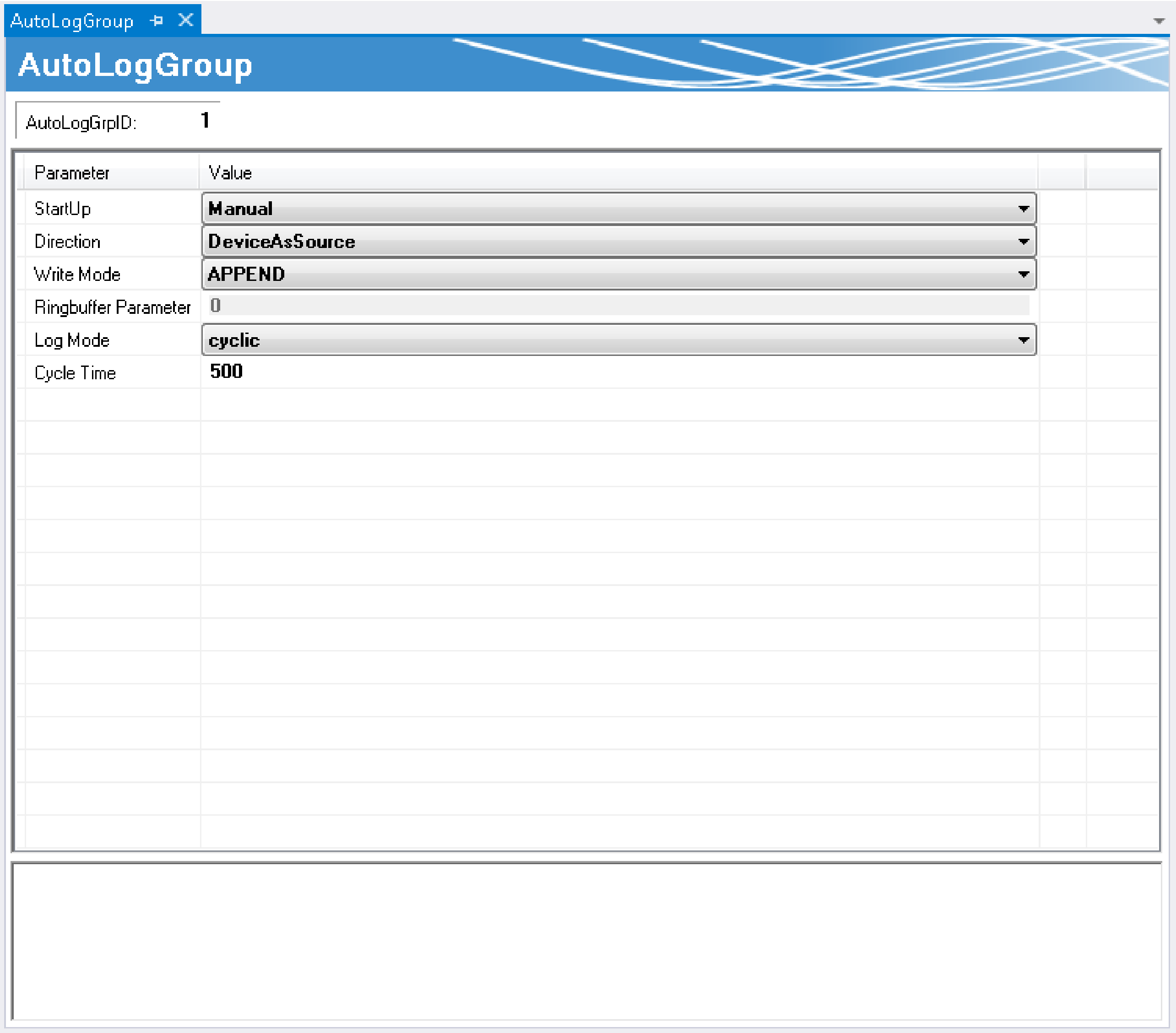


5.1.1.1.4 Configuring AutoLog groups

A new AutoLog group for the database configuration can be added via the command **Add New AutologGroup** in the context menu of a database configuration or via the toolbar. These AutoLog groups refer to the parent database.



A new AutoLog group and the corresponding components are added as files to the project folder and integrated in the project. They include the ADS device, the symbol groups and the table settings. In order to save these files in the project, you should save the TwinCAT Connectivity project file. The files can then be edited in editors or in the Properties window.



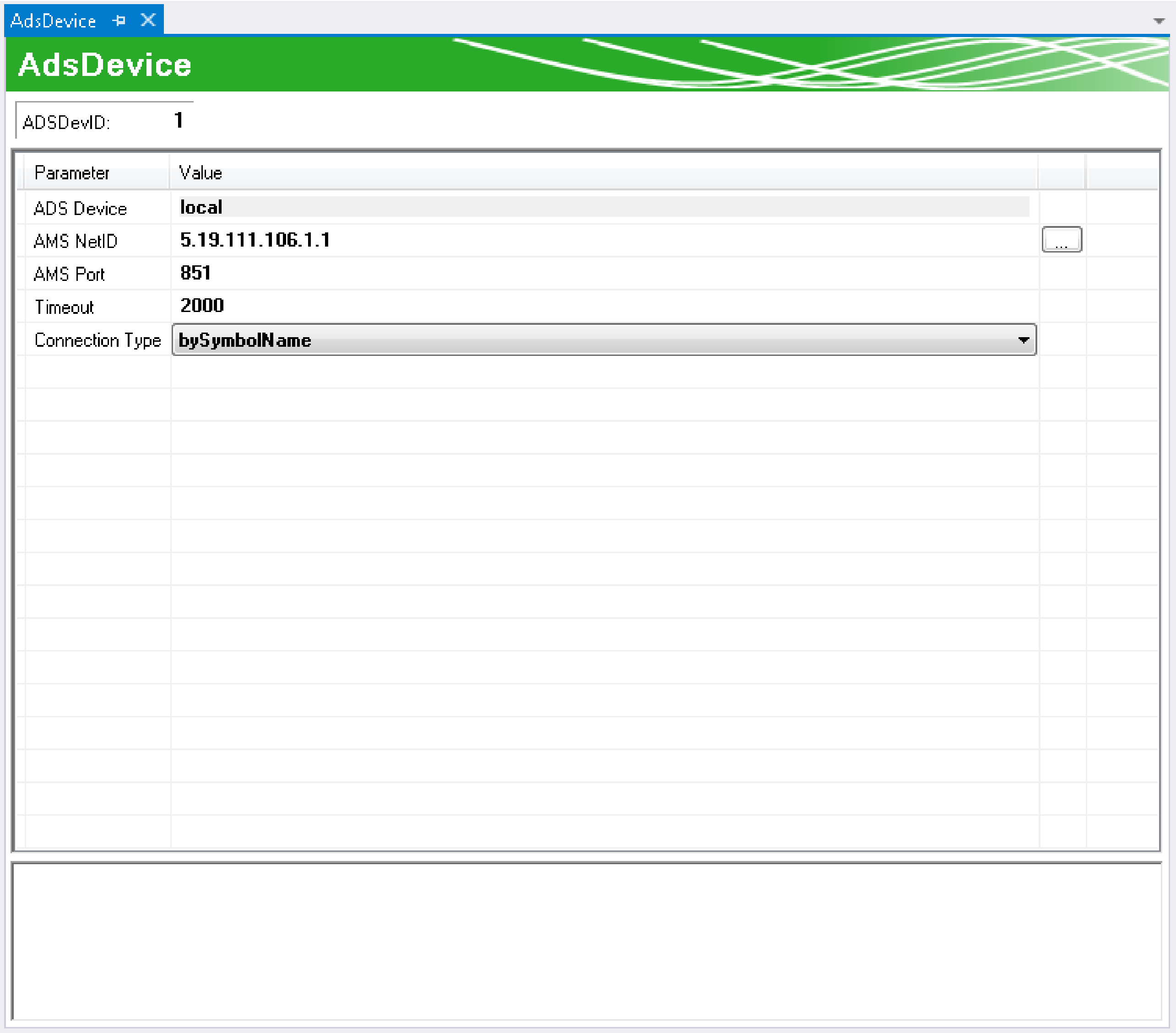
AutoLogGrpID: 1

Parameter	Value
StartUp	Manual
Direction	DeviceAsSource
Write Mode	APPEND
Ringbuffer Parameter	0
Log Mode	cyclic
Cycle Time	500

StartUp	AutoLog mode can be enabled manually (with a command in the PLC or from the configurator) or automatically during system startup.
Direction	The set ADS device is used as data target or data source.
Write mode	The data can appended in a database line-by-line, held in a ring buffer on a temporal or quantitative basis, or simply be updated at the corresponding position.
Ring buffer parameter	Depending on the setting this parameter represent the time or the cycles after which the ring buffer is updated.
Log mode	The variable is written either after a certain cycle time or when a change occurs.
Cycle Time	Cycle time after which the variable is written.

Configuring the ADS device

The ADS device is automatically created under an AutoLog group. In the most frequent use case the ADS device is the PLC runtime. The following parameters can be set in the editor:



AdsDevice

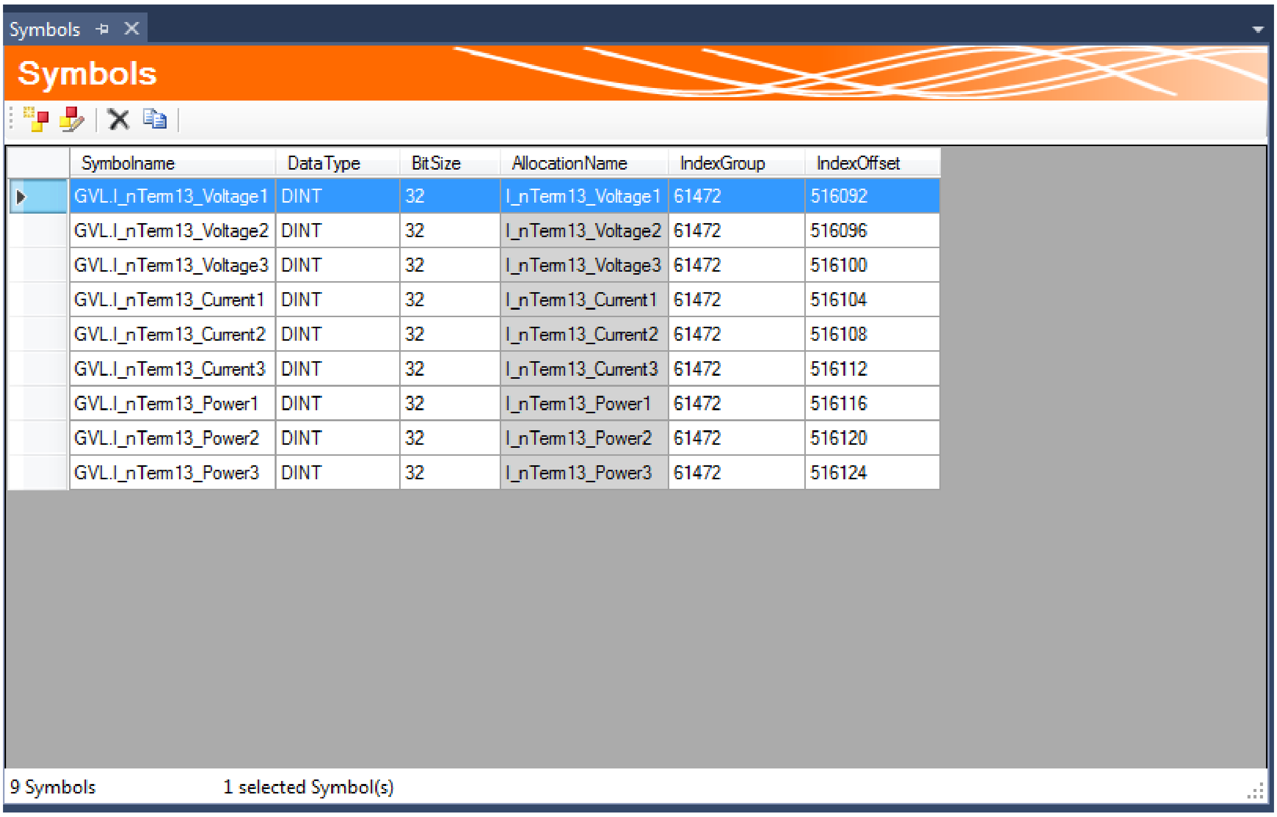
ADSDevID: 1

Parameter	Value
ADS Device	local
AMS NetID	5.19.111.106.1.1
AMS Port	851
Timeout	2000
Connection Type	bySymbolName

ADS Device	Name of the ADS target device.
AMS NetID	Address of the target device in the TwinCAT network.
AMS Port	Port of the target device in the TwinCAT network.
Timeout	Time after which it is assumed that the connection to the target device is lost.
Connection Type	bySymbolName: Connection is established based on the symbol name. byIndexGroup: Connection is established based on the memory index.

Configuring symbols

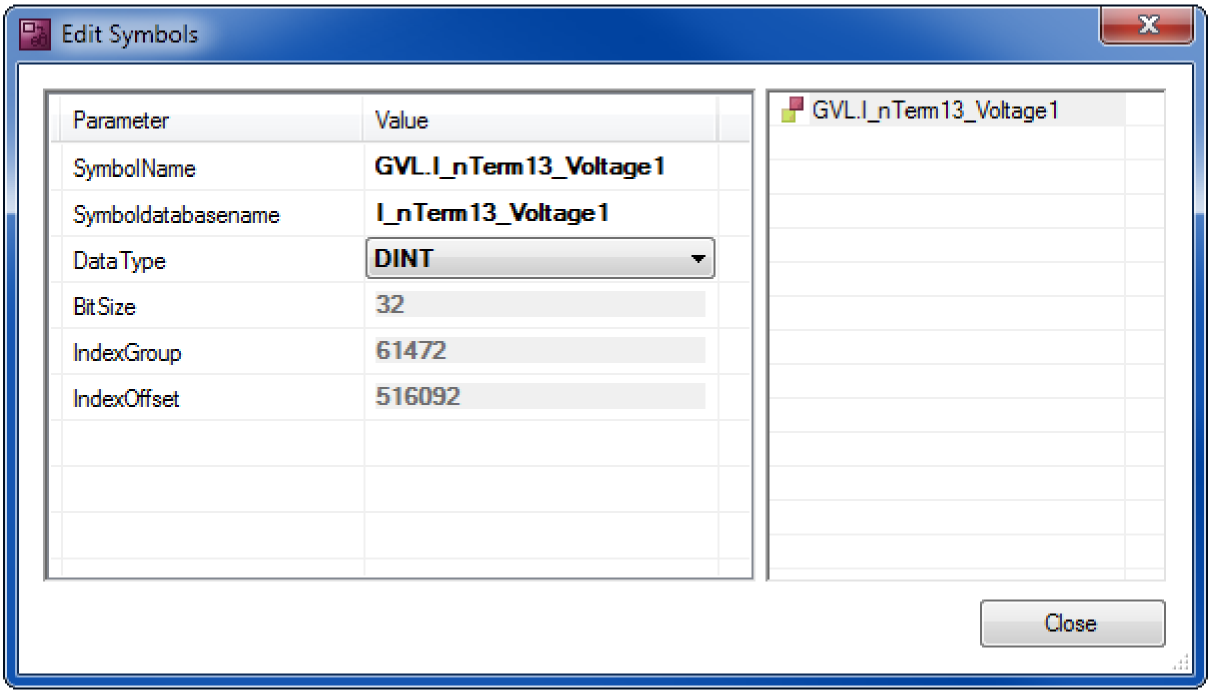
The symbols you set here are written to or read from the database, depending on whether the ADS device is the data target or the data source. The TwinCAT Target browser can be used for convenient access. Here you can search for the symbols on the target and communicate between the two tools via drag & drop.



	Symbolname	DataType	BitSize	AllocationName	IndexGroup	IndexOffset
▶	GVL.I_nTerm13_Voltage1	DINT	32	I_nTerm13_Voltage1	61472	516092
	GVL.I_nTerm13_Voltage2	DINT	32	I_nTerm13_Voltage2	61472	516096
	GVL.I_nTerm13_Voltage3	DINT	32	I_nTerm13_Voltage3	61472	516100
	GVL.I_nTerm13_Current1	DINT	32	I_nTerm13_Current1	61472	516104
	GVL.I_nTerm13_Current2	DINT	32	I_nTerm13_Current2	61472	516108
	GVL.I_nTerm13_Current3	DINT	32	I_nTerm13_Current3	61472	516112
	GVL.I_nTerm13_Power1	DINT	32	I_nTerm13_Power1	61472	516116
	GVL.I_nTerm13_Power2	DINT	32	I_nTerm13_Power2	61472	516120
	GVL.I_nTerm13_Power3	DINT	32	I_nTerm13_Power3	61472	516124

9 Symbols 1 selected Symbol(s)

Symbols can also be added manually to symbol groups or edited. The information that is required varies, depending on whether in the ADS device the connection type was selected via the symbol name or the index groups. The starting point is always the ADS device.



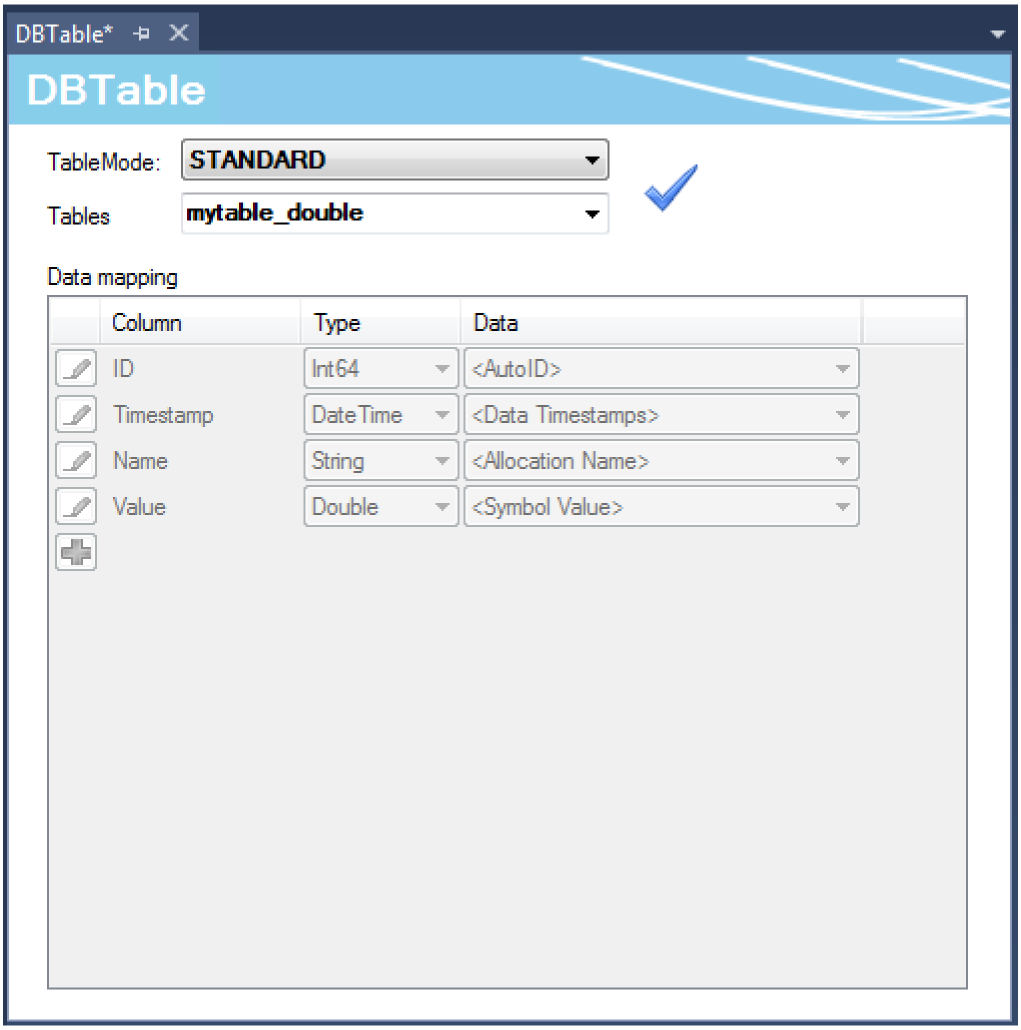
Parameter	Value
SymbolName	GVL.I_nTerm13_Voltage1
Symboldatabasename	I_nTerm13_Voltage1
DataType	DINT
BitSize	32
IndexGroup	61472
IndexOffset	516092

SymbolName	The symbol is addressed based on the set ADS device
Symbol database name	Name of the variable in the database table
DataType	PLC data type of the symbol
BitSize	Bit size of the symbols (set automatically for the data types)
IndexGroup	Index group in the TwinCAT system
IndexOffset	Index offset in the TwinCAT system

Configuring a table

The table in a database can be based on a standard table structure or on an individual structure.

The corresponding table can be selected from a list of possible tables. If the table does not yet exist, you can create it via the SQL Query Editor [▶ 43]. If you select the standard table structure, a blue tick indicates whether the selected table corresponds to this structure.



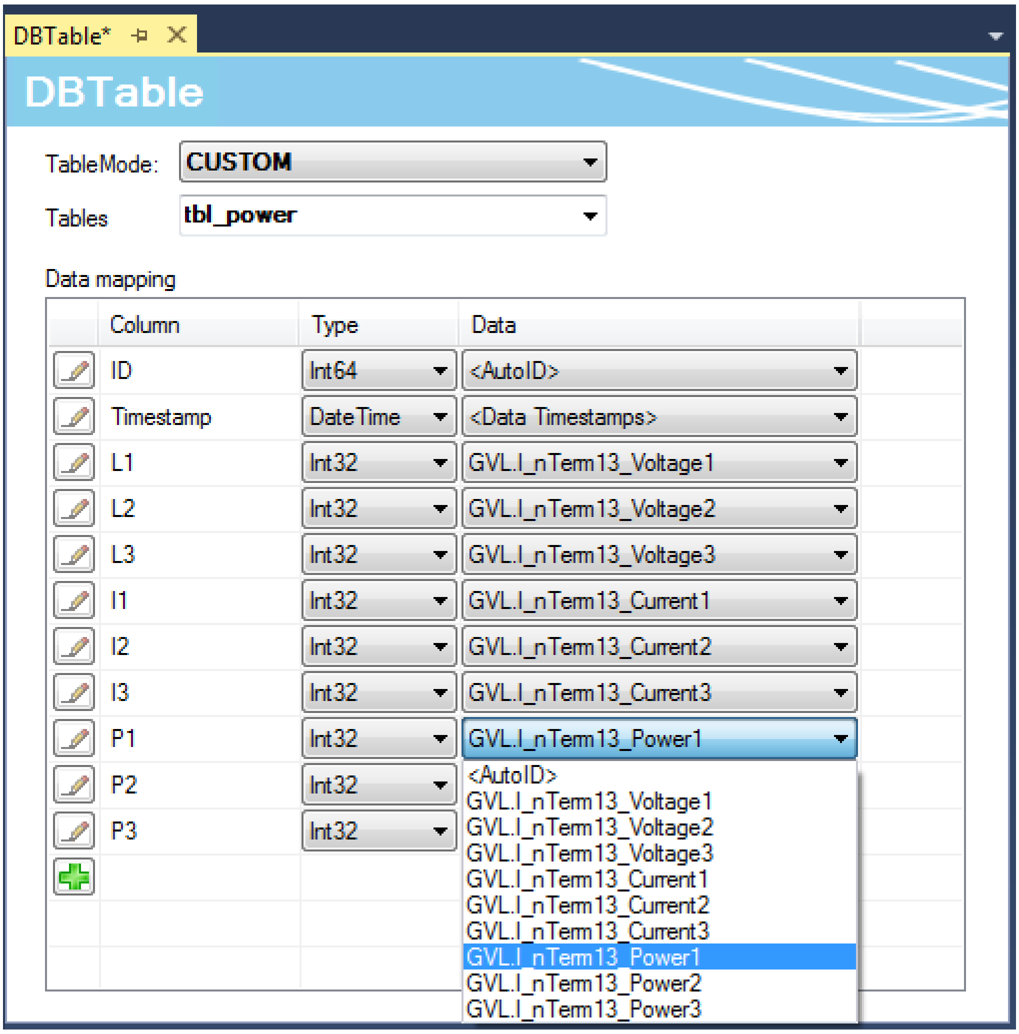
TableMode: STANDARD

Tables mytable_double ✓

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
Name	String	<Allocation Name>
Value	Double	<Symbol Value>

The specific table type offers the option to distribute the individual symbols that were set in the symbol group to the table columns in the database as required. When a record is written to the database in AutoLog mode, the current values of the symbol group at the sampling time are saved in the corresponding table column.



TableMode: CUSTOM

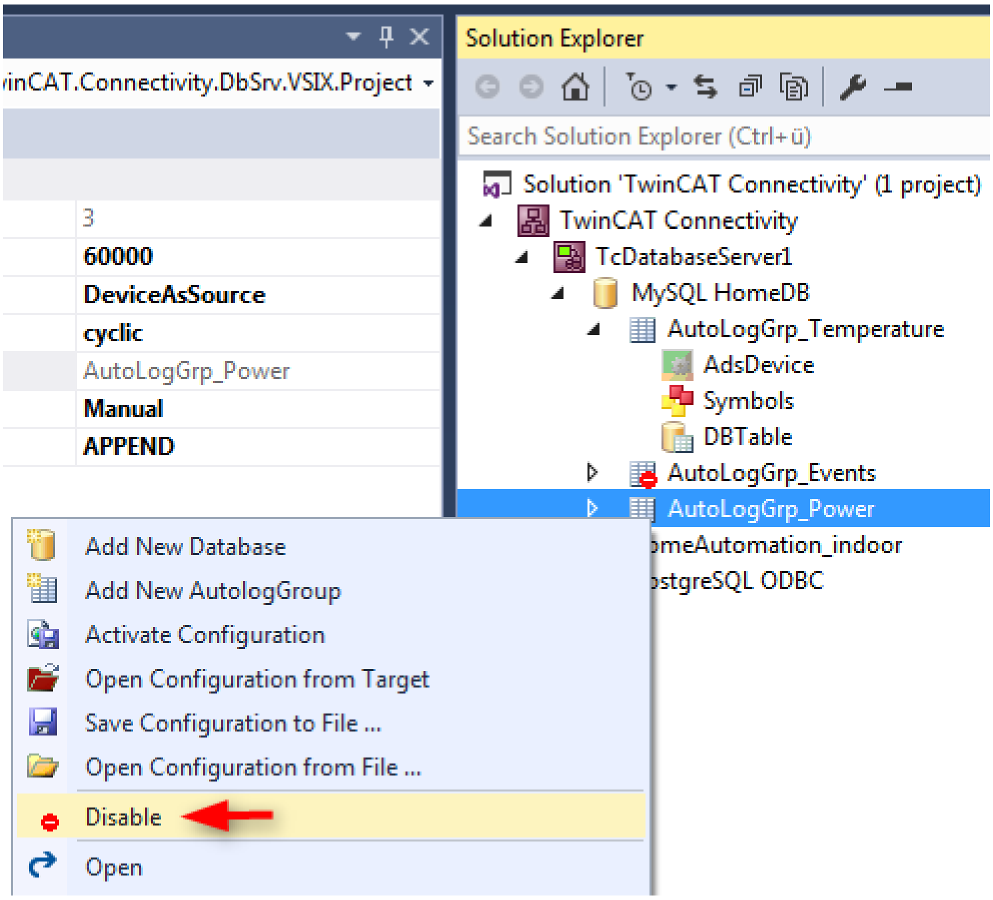
Tables tbl_power

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
L1	Int32	GVL.I_nTerm13_Voltage1
L2	Int32	GVL.I_nTerm13_Voltage2
L3	Int32	GVL.I_nTerm13_Voltage3
I1	Int32	GVL.I_nTerm13_Current1
I2	Int32	GVL.I_nTerm13_Current2
I3	Int32	GVL.I_nTerm13_Current3
P1	Int32	GVL.I_nTerm13_Power1
P2	Int32	
P3	Int32	

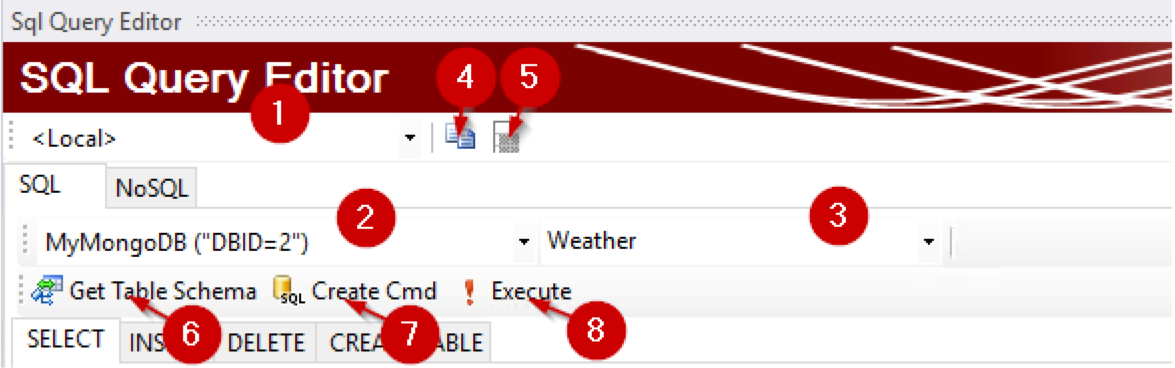
Disabling AutoLog groups

Just like individual database configurations, individual AutoLog groups can also be disabled in the project. These are then ignored when the project is enabled on the target system. A deactivated AutoLog group is indicated by a red mark. It can be reactivated with the same command.



5.1.1.1.5 SQL Query Editor

The SQL Query Editor is a Database Server tool that supports the development of your application. The tool can be used to test connections and SQL commands and to check the compatibility between PLC and databases.



ID	Name	Function
1	Target system	Choose Target System with installed TwinCAT Database Server
2	Database	Selecting the configured database connection
3	Table	Selecting the existing tables in the database
4	Copying for PLC	Copying the SQL command to the PLC string. This can be copied into the PLC source code. Special characters are automatically captured and formatted.
5	Export TC3	Exporting the table schema into a PLC structure. This can be used in the program for SQL commands, for example.
6	Get Table Schema	Reading the table structure
7	Create Cmd	Creating an SQL command, based on the table structure
8	Execute	Executing the SQL command

First select the target system from the routes of your TwinCAT system (1). The TwinCAT Database Server must be installed on the target system. If a NoSQL database is stored in the configuration, an additional NoSQL tab is visible. You will find the documentation in a subitem below.

All configured databases (2) are displayed, once you have activated the database configurations on the target system. You can also select one of the available tables (3) from the database. Based on this table, you can generate SQL commands from the SQL Query Editor and send them to the database. The SQL commands have different syntax, depending on database type.

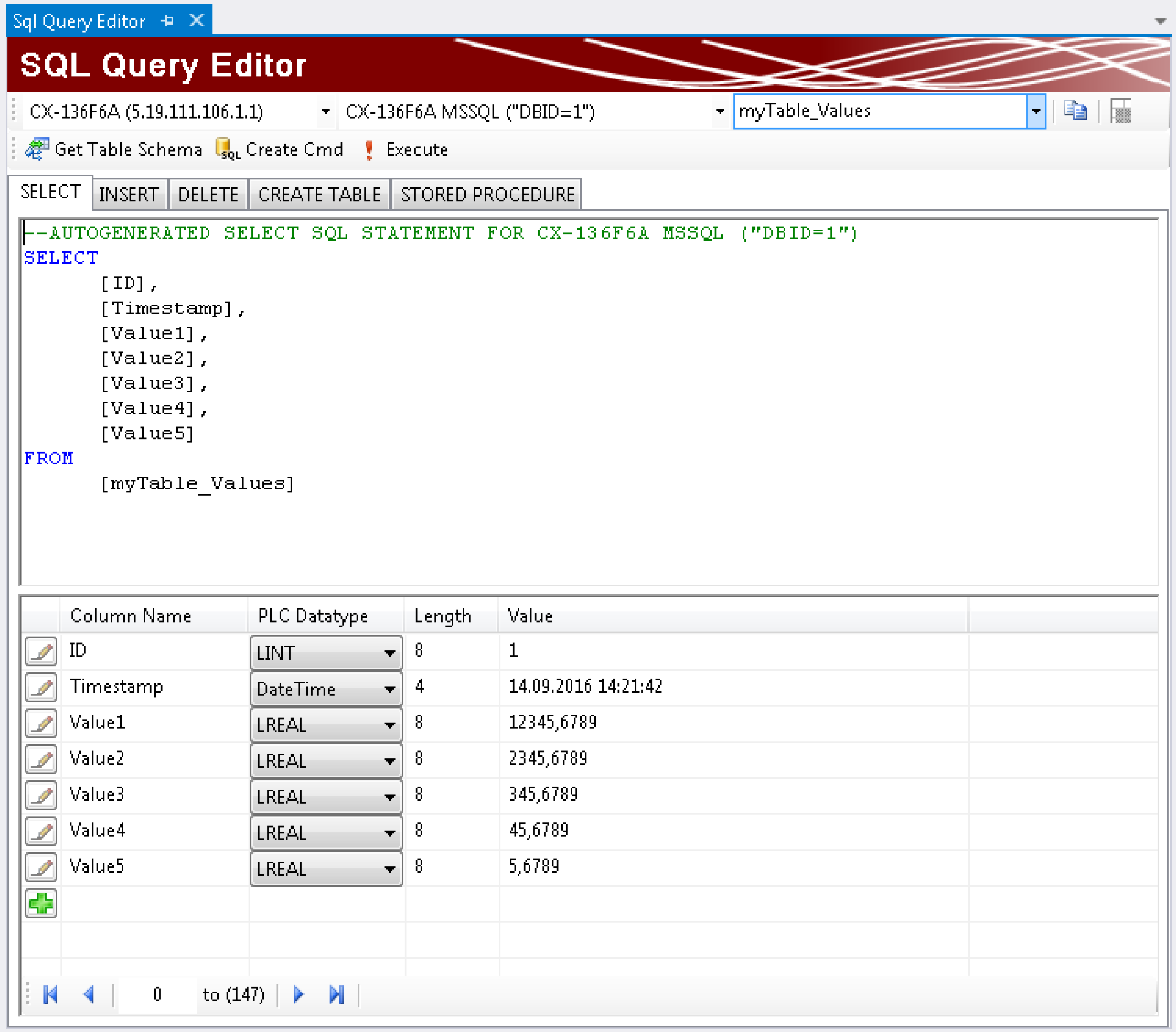
Three commands are available for generating the individual SQL commands:

- Get Table Schema: Calls up the structure of the selected table.
    - Information such as the column name, PLC data type and size of variables is displayed. The retrieved structure can also be prepared for your PLC application via the commands **Copy for PLC** (4) or **Export TC3** (5).
- Create Cmd: An SQL command is generated in the command text box, depending on the selected tab. The command syntax may differ, depending on the database type. The previously read table schema is used here.
    - The created SQL command can optionally be modified.
- Execute: The SQL command shown in the text box is executed and returns values, if applicable.

The differences in the individual SQL commands are explained below.

Select command

Select commands can be created and sent via the **SELECT** tab. Select commands give the opportunity to read records from the databases. After executing the command, values are returned if they exist in the table. They are listed under "Value" in the table structure display. Use the arrows under the display to navigate through the individual records.



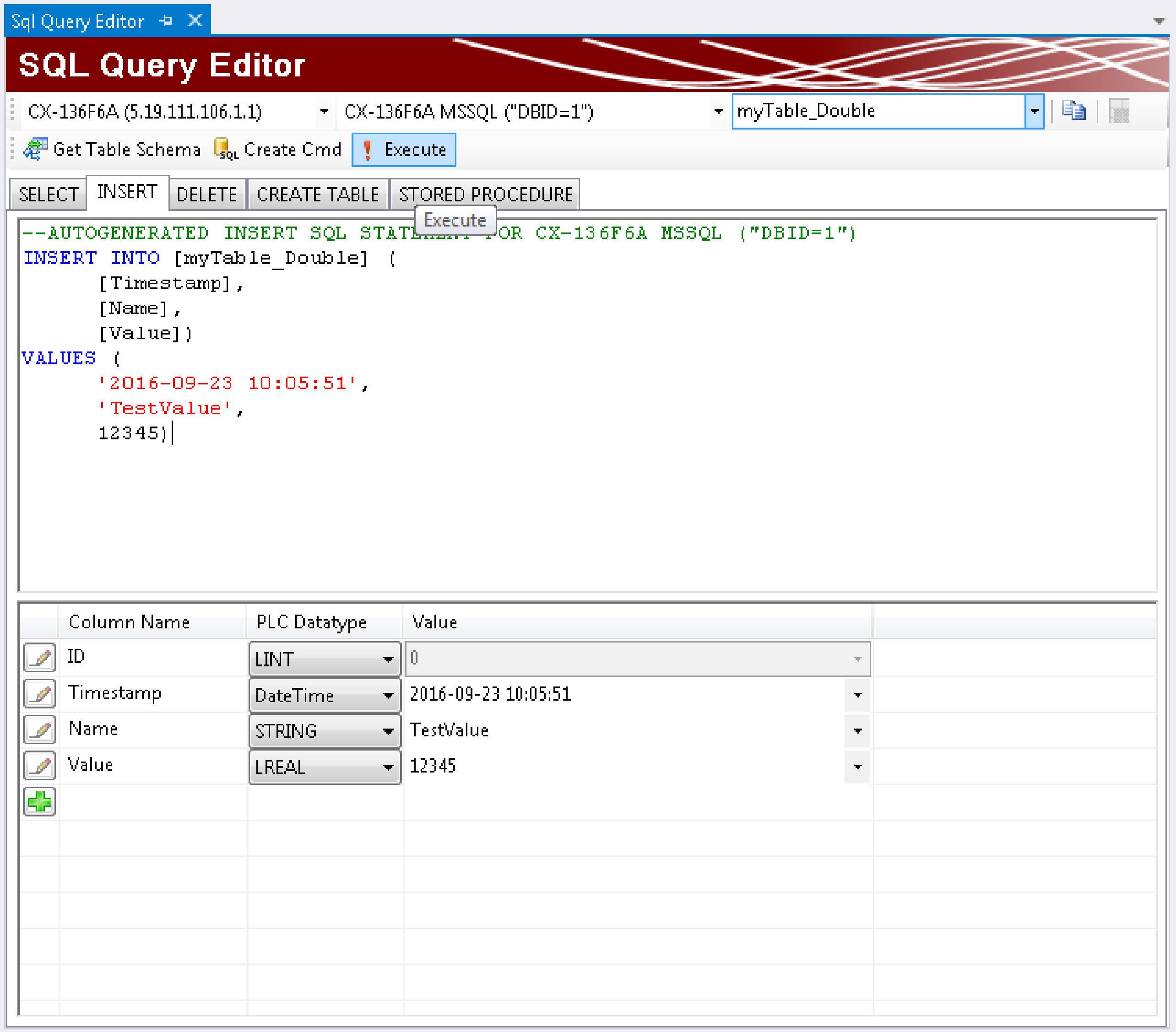
```
--AUTOGENERATED SELECT SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
SELECT
        [ID],
        [Timestamp],
        [Value1],
        [Value2],
        [Value3],
        [Value4],
        [Value5]
FROM
        [myTable_Values]
```

Column Name	PLC Datatype	Length	Value
ID	LINT	8	1
Timestamp	DateTime	4	14.09.2016 14:21:42
Value1	LREAL	8	12345,6789
Value2	LREAL	8	2345,6789
Value3	LREAL	8	345,6789
Value4	LREAL	8	45,6789
Value5	LREAL	8	5,6789

Insert command

The Insert command gives the opportunity to write records into the table. The values under "Value" can be modified once the table structure has been retrieved. If the command is then generated, the values in the Insert command will automatically be in the right format. These values are written into the table when the command is executed.

This value cannot be customized if automatic ID generation is used.



CX-136F6A (5.19.111.106.1.1)
CX-136F6A MSSQL ("DBID=1")
myTable_Double

```
--AUTOGENERATED INSERT SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
INSERT INTO [myTable_Double] (
       [Timestamp],
       [Name],
       [Value])
VALUES (
       '2016-09-23 10:05:51',
       'TestValue',
       12345)
```

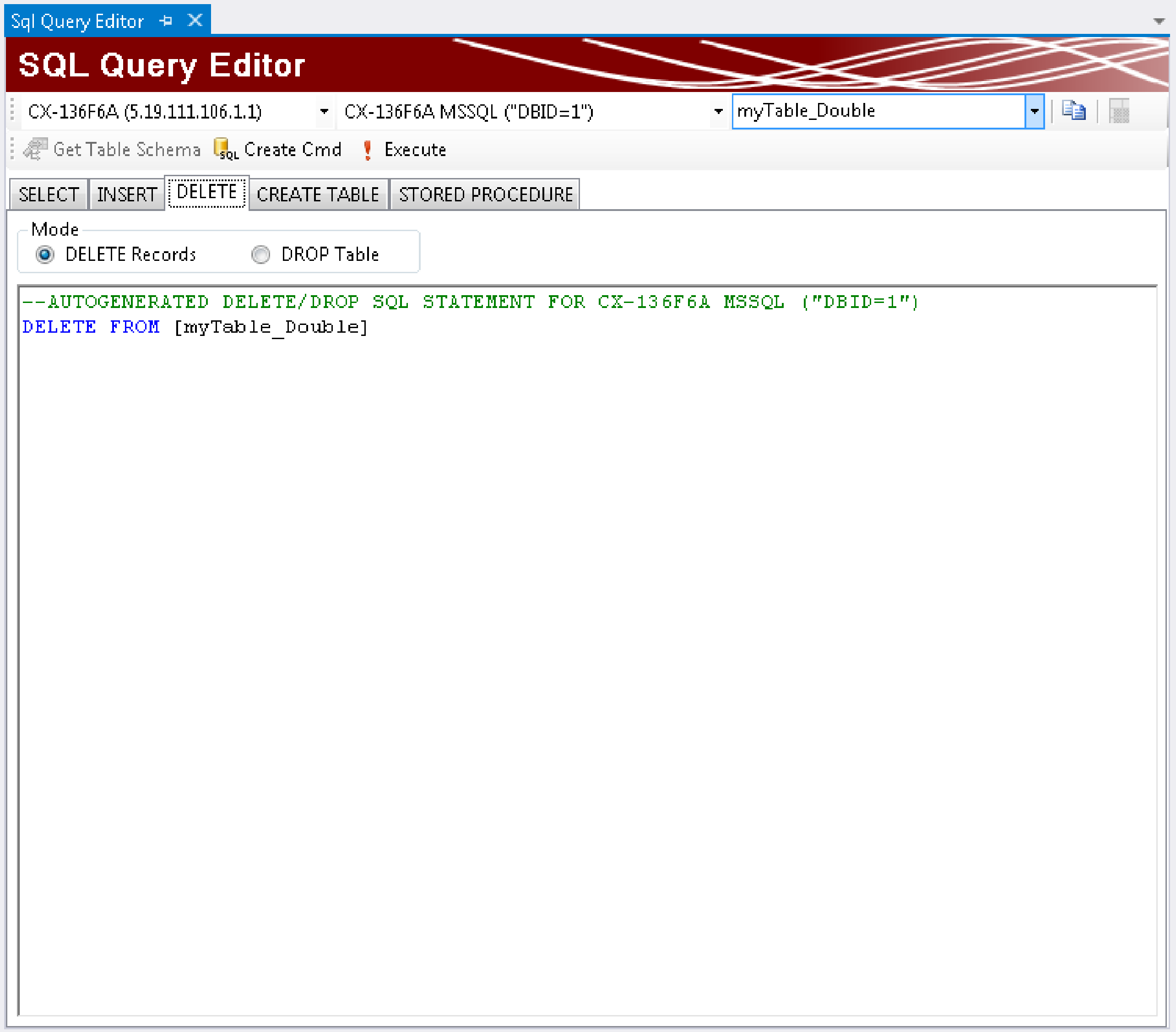
Column Name	PLC Datatype	Value
ID	LINT	0
Timestamp	DateTime	2016-09-23 10:05:51
Name	STRING	TestValue
Value	LREAL	12345

Delete command

The Delete command has two functions.

1. **DELETE Records**: Deletes the contents of a table.
2. **DROP table:** Deletes the whole table.

This SQL command can also be customized, in order to delete only a particular section of the table, for example.

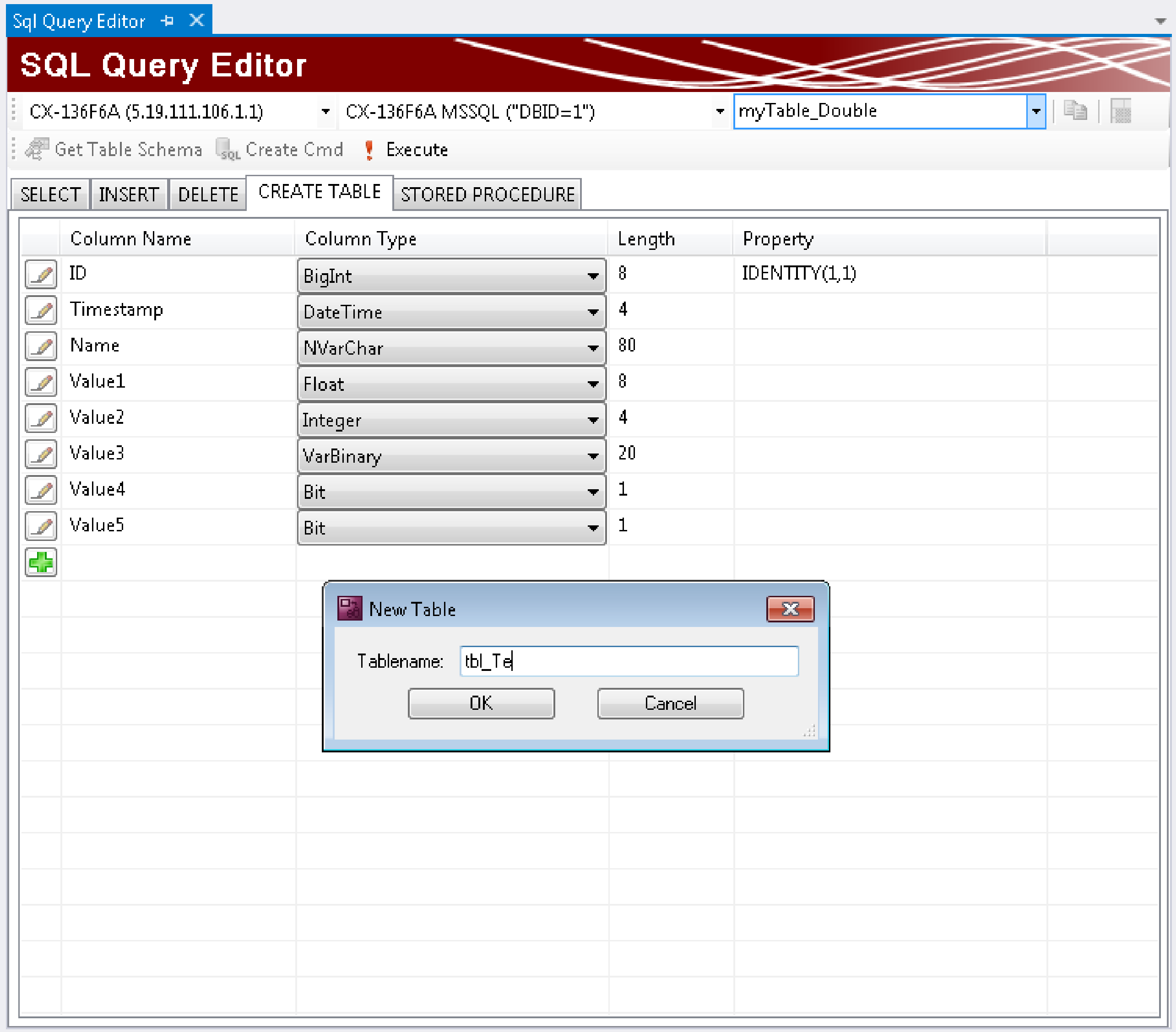


```
--AUTOGENERATED DELETE/DROP SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
DELETE FROM [myTable_Double]
```

Create Table command

The **CREATE TABLE** tab can be used to create tables within the database. Further columns can be added to the table with (+), as required. Once you have specified the column name and type, you can specify additional properties, in order to generate automatic IDs, for example.

The table name can be determined by executing the command. The table with the configured table structure is created.



Column Name	Column Type	Length	Property
ID	BigInt	8	IDENTITY(1,1)
Timestamp	DateTime	4	
Name	NVarChar	80	
Value1	Float	8	
Value2	Integer	4	
Value3	VarBinary	20	
Value4	Bit	1	
Value5	Bit	1	

New Table

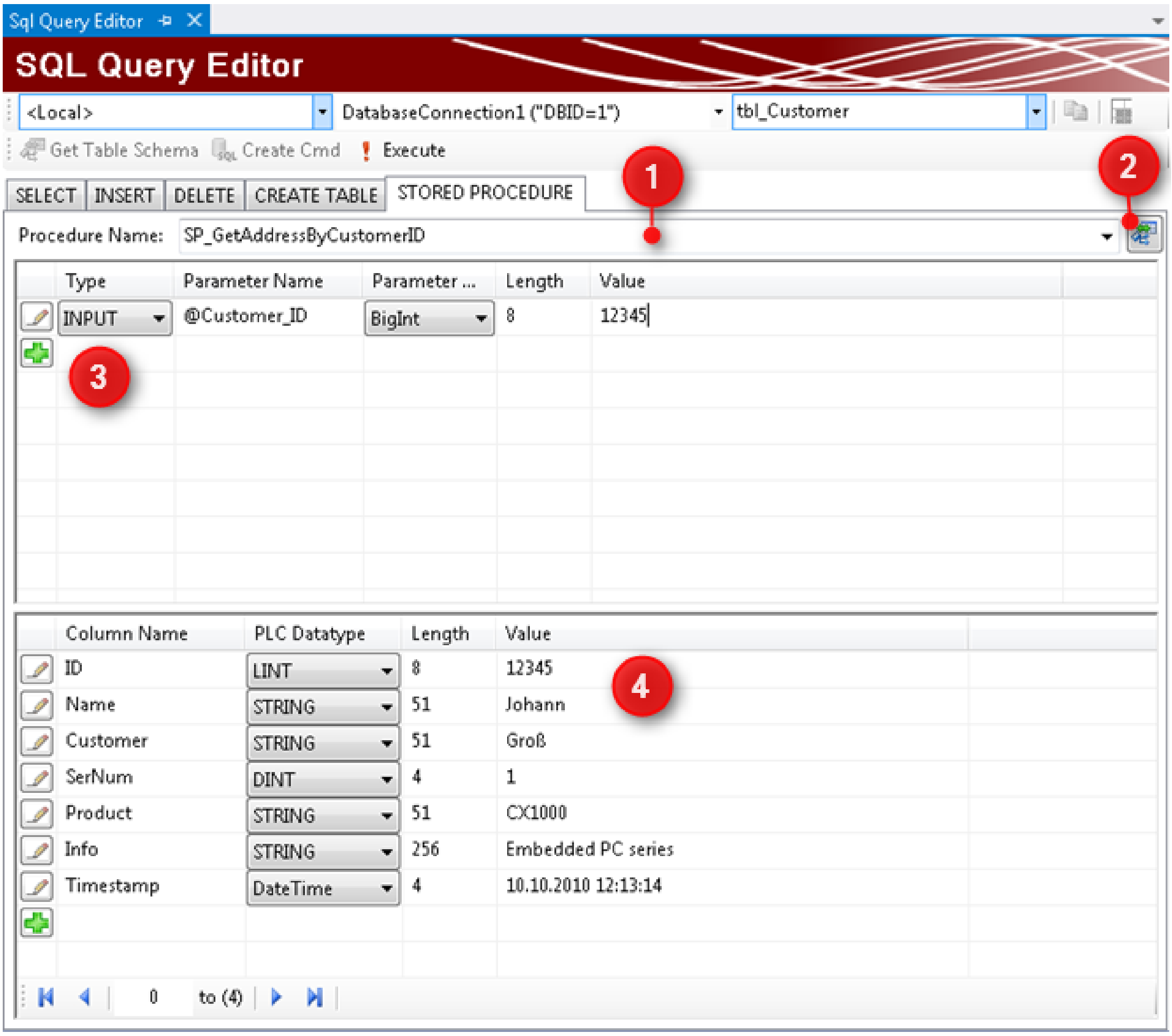
Tablename: tbl_Te

Stored Procedures

The TwinCAT Database Server supports "Stored Procedures", which provide numerous databases for processing more complex queries at the database level or to make a simplified interface available.

If Stored Procedures are available in the database and the table, you can list and select them (1). The input and output parameters can be picked up automatically (2) and transferred to the tables in the display (3)(4).

The parameter type, name and data type are displayed there. In addition you can insert values here, in order to execute the Stored Procedures with the input values via "Execute". The result is displayed in the output values (4). If several records are returned, the arrow keys can be used to switch between them. This functionality serves as development aid for the call in the PLC. The results are returned there by calling the corresponding function block [▶ 191].



<Local>   DatabaseConnection1 ("DBID=1")   tbl_Customer

Procedure Name: SP_GetAddressByCustomerID

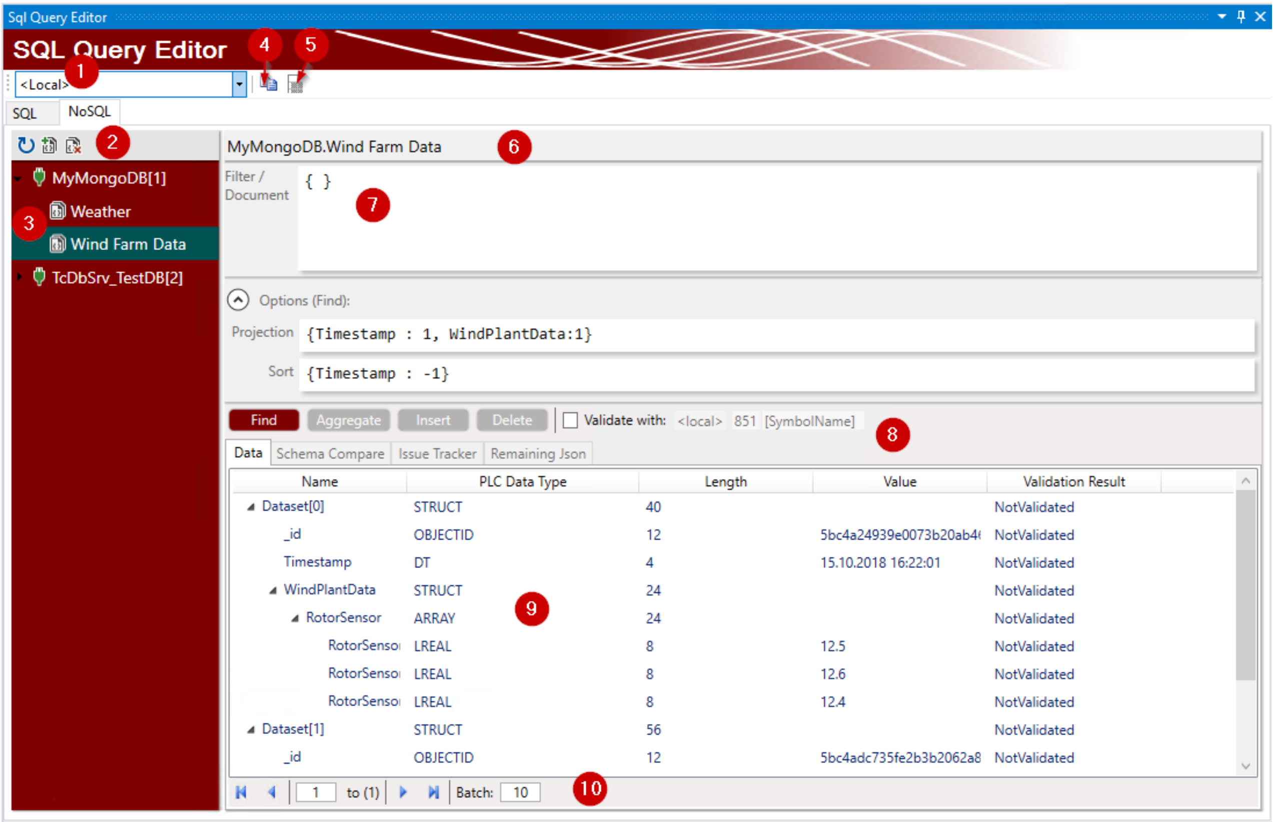
Type	Parameter Name	Parameter ...	Length	Value
INPUT	@Customer_ID	BigInt	8	12345

Column Name	PLC Datatype	Length	Value
ID	LINT	8	12345
Name	STRING	51	Johann
Customer	STRING	51	Groß
SerNum	DINT	4	1
Product	STRING	51	CX1000
Info	STRING	256	Embedded PC series
Timestamp	DateTime	4	10.10.2010 12:13:14

NoSql Query Editor

The NoSQL tab supports the special functions of NoSQL databases. It is only visible if a NoSQL database has been configured and uploaded to your target device (1).

Your available databases are then listed in the list of NoSQL databases (3). SQL databases are not displayed in this list. All existing collections are now listed under a selected database. In the menu bar (2) the list can be updated, and collections can be added or deleted.



MyMongoDB.Wind Farm Data

Filter / Document: { }

Options (Find):

Projection: {Timestamp : 1, WindPlantData:1}

Sort: {Timestamp : -1}

□ Validate with: <local> 851 [SymbolName]

Name	PLC Data Type	Length	Value	Validation Result
Dataset[0]	STRUCT	40		NotValidated
_id	OBJECTID	12	5bc4a24939e0073b20ab4(	NotValidated
Timestamp	DT	4	15.10.2018 16:22:01	NotValidated
WindPlantData	STRUCT	24		NotValidated
RotorSensor	ARRAY	24		NotValidated
RotorSenso	LREAL	8	12.5	NotValidated
RotorSenso	LREAL	8	12.6	NotValidated
RotorSenso	LREAL	8	12.4	NotValidated
Dataset[1]	STRUCT	56		NotValidated
_id	OBJECTID	12	5bc4adc735fe2b3b2062a8	NotValidated

1 to (1) Batch: 10

ID	Name	Function
1	Target system	Selecting the target system with installed TwinCAT Database Server
2	Database menu	Updating, adding and deleting collections
3	Database explorer	Selecting the existing database and collections
4	Copying for PLC	Copying the SQL command to a PLC string. This can be copied into the PLC source code. Special characters are automatically captured and formatted.
5	Export TC3	Exporting the table schema into a PLC structure. This can be used in the program for SQL commands, for example.
6	Collection path	Database name and collection that is currently selected.
7	Document/Filter	Depending on which function is used, this input field acts as a document or as a filter in JSON format. If you want to execute a Find operation and also carry out a projection or sort operation, you can fill these fields with *Options(Find)* below.
8	Control elements	Control elements for interaction with the TwinCAT Database Server.
9	Data display	List of returned data
10	Navigation	Allows iteration through the returned records

Once a collection has been selected, the target (6) of the query to be sent changes. The following functions (8) are provided:

Find: Executes a search query with the filter entered in the text field (7). Optionally, a projection or sorting operation can also be executed via the *Options(Find)* fields. Data is returned and listed in the data display (9). The syntax of the filters is database-specific.

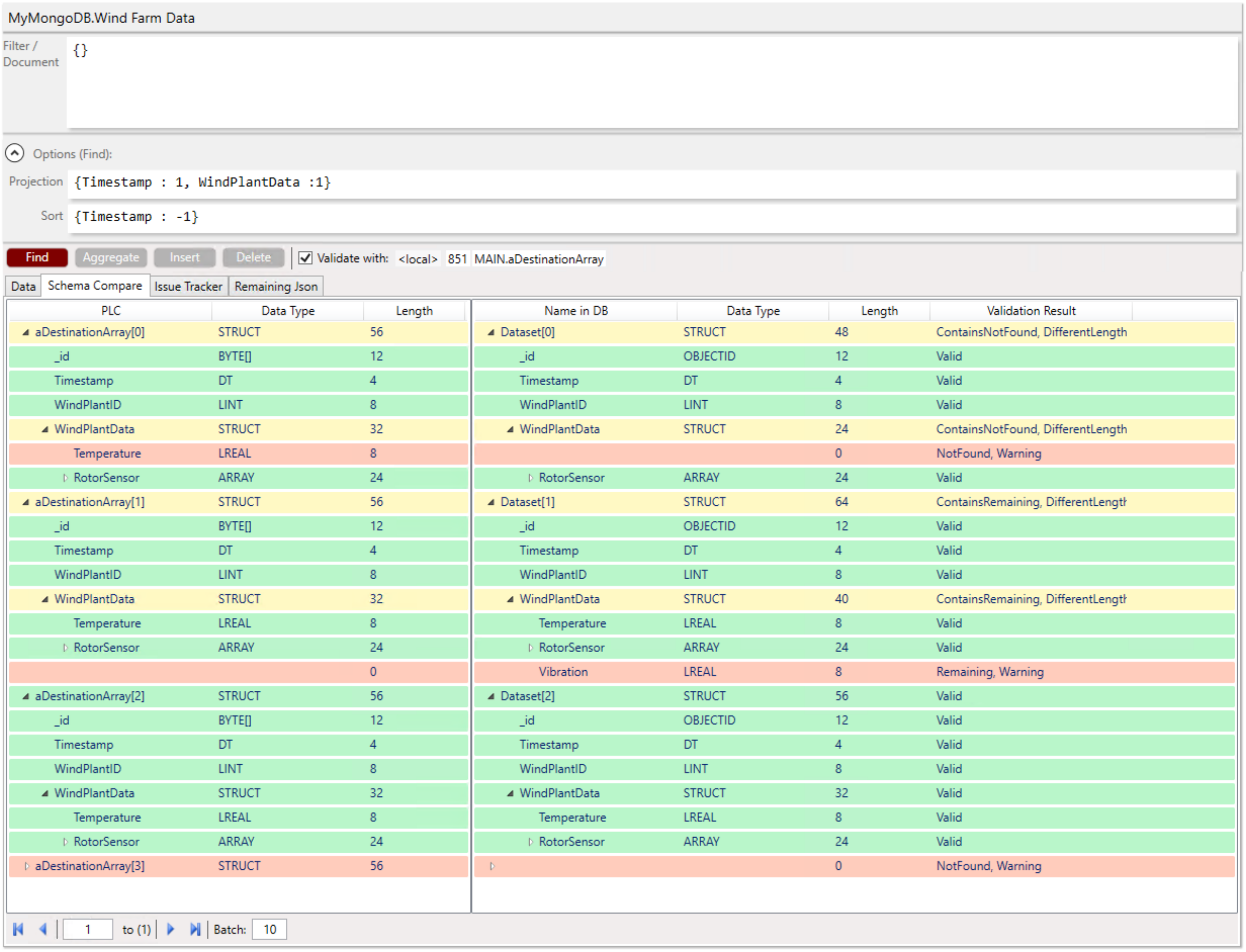
Aggregate: Executes an aggregation with the parameters entered in the text field (7). Data is returned and listed in the data display (9). The syntax of the filters is database-specific.

Insert: Executes an insert query of the (JSON) document or document array entered in the text field (7). These are then written to the collection.

Delete: Executes a delete query on the data found with the filter in the text field (7). Any data that is found is deleted from the collection.

Validate: If this option is selected, the data queries are not automatically parsed according to their own schema, but an attempt is made to map these data to the structure of the symbol from the PLC, which was specified via these parameters.

With the latter function, a Find query may lead to conflicts. In contrast to structures in the PLC process image, records in NoSQL databases do not have to follow a fixed schema. It is therefore possible that queried documents have no data relating to a specific element in the PLC structure. Or the record carries data that does not occur in the PLC structure. These data are assigned via the name or the attribute "ElementName" in the PLC.



MyMongoDB.Wind Farm Data

Filter / Document: {}

Options (Find):

Projection: {Timestamp : 1, WindPlantData :1}

Sort: {Timestamp : -1}

☑ Validate with: <local> 851 MAIN.aDestinationArray

PLC	Data Type	Length	Name in DB	Data Type	Length	Validation Result
aDestinationArray[0]	STRUCT	56	Dataset[0]	STRUCT	48	ContainsNotFound, DifferentLength
_id	BYTE[]	12	_id	OBJECTID	12	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
WindPlantID	LINT	8	WindPlantID	LINT	8	Valid
WindPlantData	STRUCT	32	WindPlantData	STRUCT	24	ContainsNotFound, DifferentLength
Temperature	LREAL	8			0	NotFound, Warning
RotorSensor	ARRAY	24	RotorSensor	ARRAY	24	Valid
aDestinationArray[1]	STRUCT	56	Dataset[1]	STRUCT	64	ContainsRemaining, DifferentLengt
_id	BYTE[]	12	_id	OBJECTID	12	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
WindPlantID	LINT	8	WindPlantID	LINT	8	Valid
WindPlantData	STRUCT	32	WindPlantData	STRUCT	40	ContainsRemaining, DifferentLengt
Temperature	LREAL	8	Temperature	LREAL	8	Valid
RotorSensor	ARRAY	24	RotorSensor	ARRAY	24	Valid
		0	Vibration	LREAL	8	Remaining, Warning
aDestinationArray[2]	STRUCT	56	Dataset[2]	STRUCT	56	Valid
_id	BYTE[]	12	_id	OBJECTID	12	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
WindPlantID	LINT	8	WindPlantID	LINT	8	Valid
WindPlantData	STRUCT	32	WindPlantData	STRUCT	32	Valid
Temperature	LREAL	8	Temperature	LREAL	8	Valid
RotorSensor	ARRAY	24	RotorSensor	ARRAY	24	Valid
aDestinationArray[3]	STRUCT	56			0	NotFound, Warning

1 to (1) Batch: 10

The differences in the data can be examined via the "Schema Compare" tab. The above sample shows that the variable "*Temperature"* of data type *LREAL* has been created in the PLC structure "*WindPlantData*" for the first document returned. However, the read record has no data for this variable. In the second document the variable *"Vibration"* is missing in the PLC. The corresponding colors show the weighting of the conflict:

Red: too many or too few data available.
Yellow: The byte length of the record does not match, or underlying records are left over or missing.
Green: No conflicts

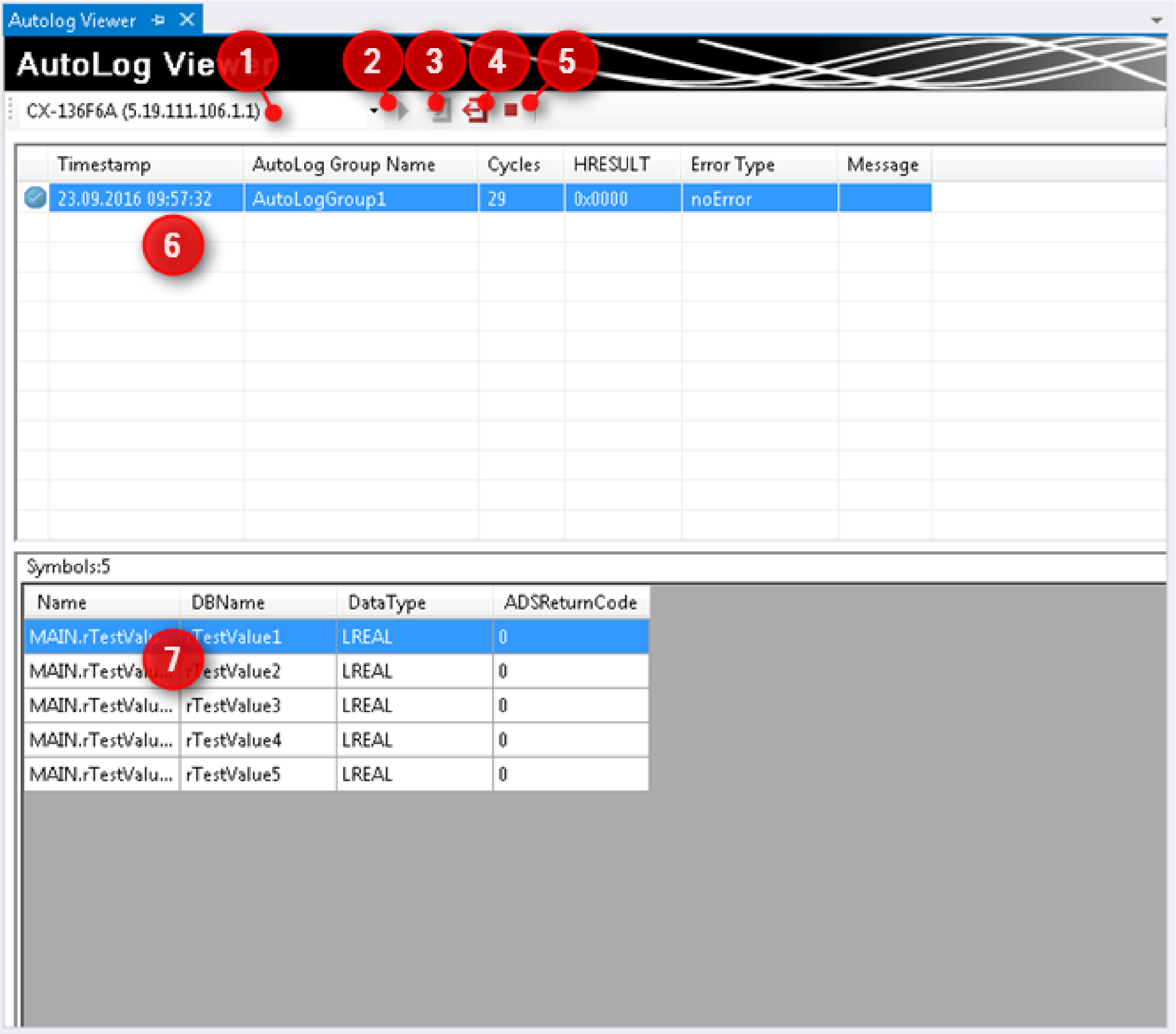
These conflicts are also listed under the "Issue Tracker" tab. It can also be read into the PLC as a string array, if required.

The "Remaining JSON" tab returns any remaining records as JSON. This information can also be read into the PLC as a string.

The control elements in the status bar can be used to iterate through the data, similar to the SQL tab. The number of records displayed simultaneously can be specified.

5.1.1.1.6 AutoLog view

The AutoLog Viewer of the TwinCAT Database Server is a tool for controlling and monitoring the AutoLog mode. You can log into a target system, similar to the TwinCAT PLC. In logged-in state the AutoLog mode can be started or stopped. Information on the current state of the logging is shown in the lower part of the window. When an AutoLog group is selected, further information is displayed via the logged symbols.



CX-136F6A (5.19.111.106.1.1)

Timestamp	AutoLog Group Name	Cycles	HRESULT	Error Type	Message
23.09.2016 09:57:32	AutoLogGroup1	29	0x0000	noError	

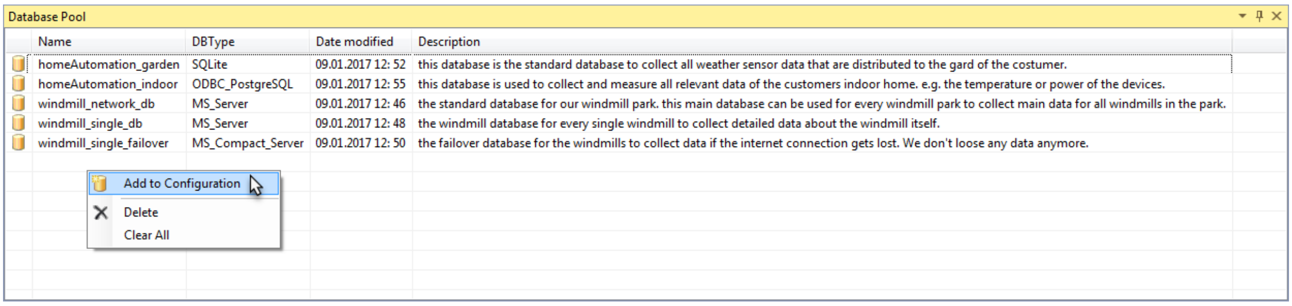
Symbols:5

Name	DBName	DataType	ADSReturnCode
MAIN.rTestVal	TestValue1	LREAL	0
MAIN.rTestVa	estValue2	LREAL	0
MAIN.rTestValu...	rTestValue3	LREAL	0
MAIN.rTestValu...	rTestValue4	LREAL	0
MAIN.rTestValu...	rTestValue5	LREAL	0

ID	Name	Function
1	Target system	Choose Target System with installed TwinCAT Database Server
2	Start	Manual start of the AutoLog mode
3	Login	Logging into the active AutoLog process
4	Logout	Logging out of the active AutoLog process
5	Stop	Manual stop of the AutoLog mode
6	AutoLog groups	List of configured AutoLog groups on the target system
7	Symbols	List of configured symbols for the selected AutoLog group

5.1.1.1.7 Database configuration pool

The database configuration pool is a global repository for database configurations on the development system. It is used by developers as storage location for project-independent database configurations or templates for repeatedly used configurations. This pool uses the same user-specific storage location for the Visual Studio integration and for the standalone configurator. The files are retained when the TwinCAT Database Server is uninstalled.

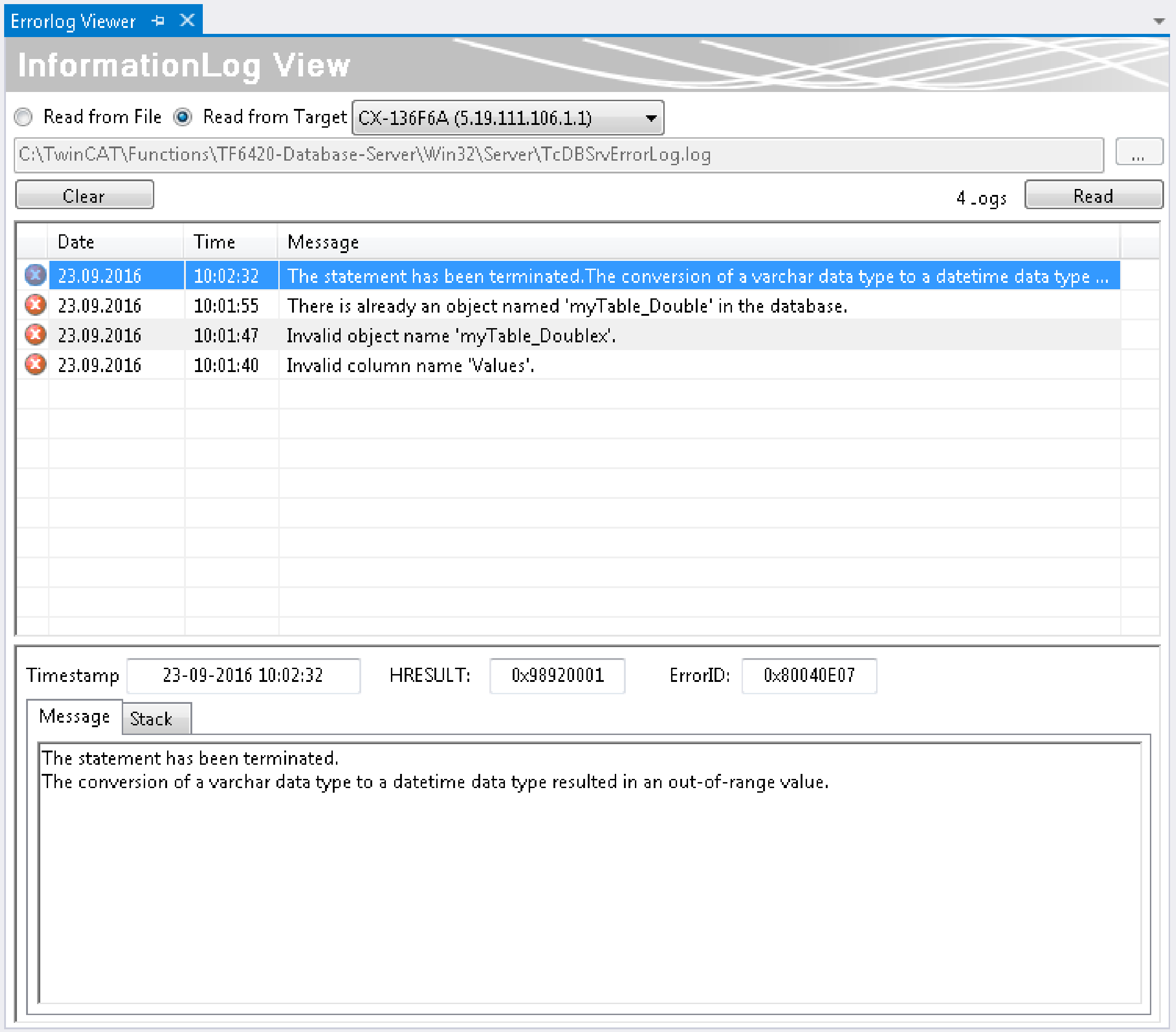


Name	DBType	Date modified	Description
homeAutomation_garden	SQLite	09.01.2017 12: 52	this database is the standard database to collect all weather sensor data that are distributed to the gard of the costumer.
homeAutomation_indoor	ODBC_PostgreSQL	09.01.2017 12: 55	this database is used to collect and measure all relevant data of the customers indoor home. e.g. the temperature or power of the devices.
windmill_network_db	MS_Server	09.01.2017 12: 46	the standard database for our windmill park. this main database can be used for every windmill park to collect main data for all windmills in the park.
windmill_single_db	MS_Server	09.01.2017 12: 48	the windmill database for every single windmill to collect detailed data about the windmill itself.
windmill_single_failover	MS_Compact_Server	09.01.2017 12: 50	the failover database for the windmills to collect data if the internet connection gets lost. We don't loose any data anymore.

5.1.1.1.8 InformationLog View

InformationLog View is a tool for reading log files from the TwinCAT Database Server. Recorded information is displayed with a timestamp, IDs and error messages in plain text.

The log files can not only be viewed or emptied via direct file access, but also directly via the target. This is particularly advantageous with distributed Database Servers in a network, for quick and easy access to the log file. For this access a route to the target device must exist.



○ Read from File ● Read from Target CX-136F6A (5.19.111.106.1.1)

C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server\TcDBSrvErrorLog.log

4 Logs

Date	Time	Message
23.09.2016	10:02:32	The statement has been terminated.The conversion of a varchar data type to a datetime data type ...
23.09.2016	10:01:55	There is already an object named 'myTable_Double' in the database.
23.09.2016	10:01:47	Invalid object name 'myTable_Doublex'.
23.09.2016	10:01:40	Invalid column name 'Values'.

Timestamp 23-09-2016 10:02:32 HRESULT: 0x98920001 ErrorID: 0x80040E07

The statement has been terminated.
The conversion of a varchar data type to a datetime data type resulted in an out-of-range value.

5.1.1.1.9 Support Information Report

The Support Information Report is a tool for collecting product information for submission to Beckhoff technical support. Collecting product-related data such as TwinCAT version/build, product version, image version and device type reduces email traffic significantly and enables more efficient advice.

Plug-in mechanism

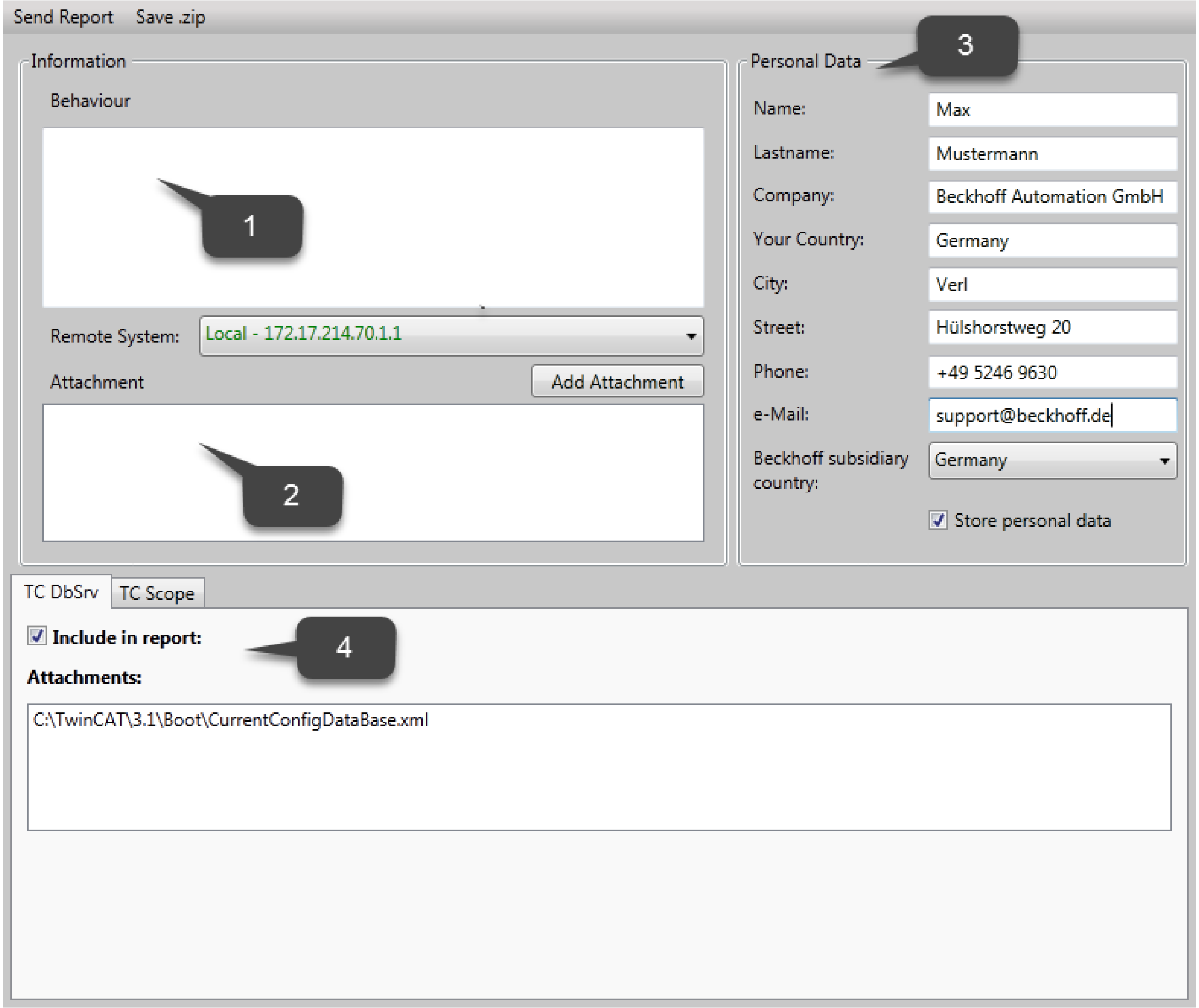
Various Beckhoff products interface with the Support Information Report via a plug-in mechanism. These products, such as the TwinCAT Database Server, have a Support Information Report entry in the corresponding product menu.

Creating and submitting a Support Information Report

✓ A Support Information Report is open.

1. Use the **Behaviour** text field to describe the behavior that occurred in as much detail as possible.
2. In the **Attachment** area, you can add files (screenshots etc.) to the report via the **Add Attachment** button, if required. Files can optionally be selected via remote access. To do this, select a target from the **Remote System** dropdown list. Depending on the selected target, it may be possible to browse Windows CE devices.
3. Enter your contact details and select a Beckhoff subsidiary for your country.
   This information is obligatory for submitting the Support Information Report.

4. You will be offered the option to store your contact details for future Support Information Reports. To do this, tick the **Store personal data** check box.
5. The product-specific plug-ins can be found in the lower section of the Support Information Report. Tick the **Include in report** check box. The information required for the product is added automatically, if it is available. The screenshot shows the current configuration of a TwinCAT Database Server in the form of an XML file as an example.
6. Submitting the Support Information Report:
   - If the device has an email connection, you can submit the Support Information Report directly to the Beckhoff subsidiary for your country via the **Send Report** button.
   - If the device does not have an email connection, you can save the Support Information Report locally as a .zip file via the **Save .zip** button and then make it available via FTP, USB etc.



Remote System: Local - 172.17.214.70.1.1

Personal Data

Name:	Max
Lastname:	Mustermann
Company:	Beckhoff Automation GmbH
Your Country:	Germany
City:	Verl
Street:	Hülshorstweg 20
Phone:	+49 5246 9630
e-Mail:	support@beckhoff.de
Beckhoff subsidiary country:	Germany

☑ Store personal data

☑ **Include in report:**

Attachments:

C:\TwinCAT\3.1\Boot\CurrentConfigDataBase.xml

5.1.1.2 Configure mode

This chapter is a compilation of all the information required for using the Configure mode of the TwinCAT Database Server. It deals with the following topics:

- Creating a project
- Creating and setting up a database configuration
- Creating and setting up AutoLog groups
- Activating a Database Server project
- Monitoring and controlling automatic logging

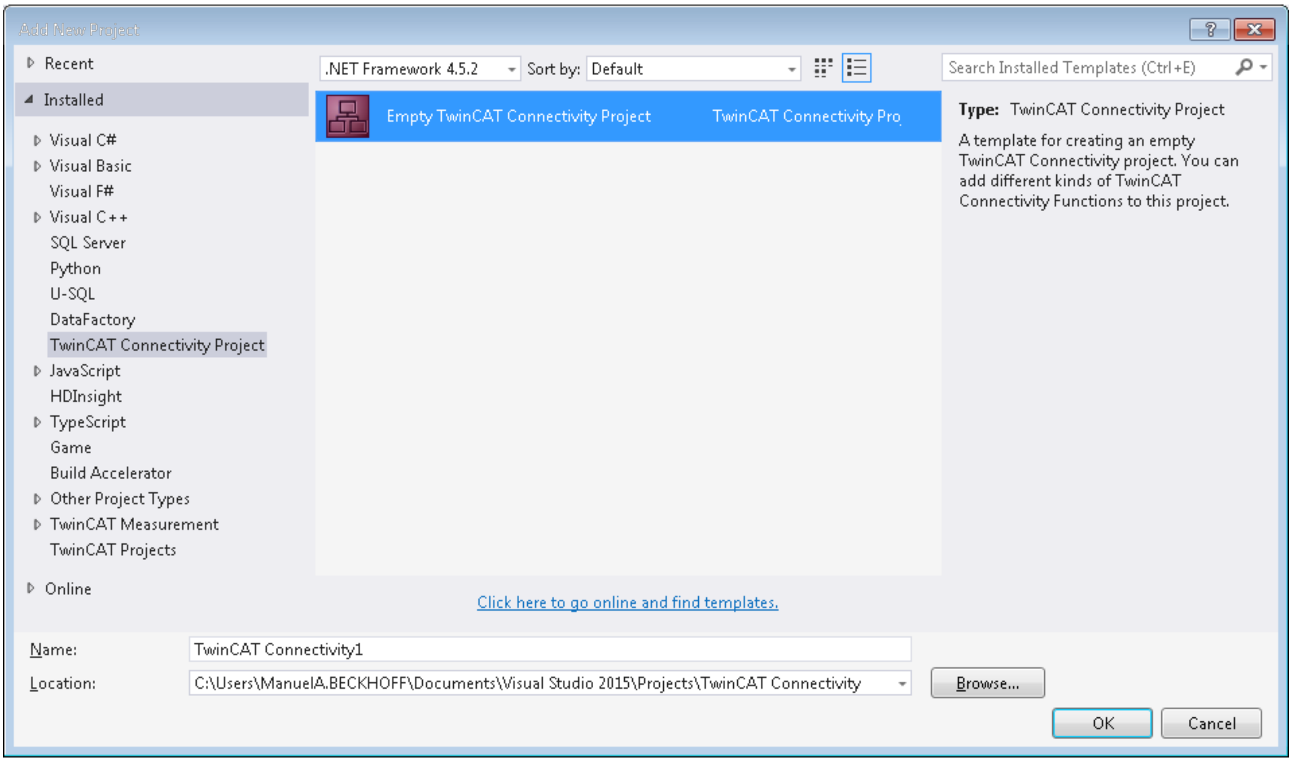
Configure Mode

In Configure mode, the bulk of the work is done in the configurator. The configuration has to be set up for the required database and for the AutoLog group. The target browser can be used for configuring the AutoLog group, for online access to a target system, and for selecting the variables to be communicated. If the **AutoStart** option is used, the communication with the configured database is established directly when TwinCAT system starts up. If the **Manual** option is selected, the communication has to be enabled via the function block FB_PLCDBAutoLog [▸ 149] or for AutoLog view.

Build Project

The TwinCAT Connectivity extension for Visual Studio provides a new project template. When a new project is created, the **TwinCAT Connectivity Project** category appears as an option.

To create a new TwinCAT Connectivity project, select **Empty TwinCAT Connectivity Project**, specify the project name and the storage location and click **OK** to add it to the solution. In this way, TwinCAT Connectivity projects or TwinCAT Database Server projects can conveniently be created in parallel with TwinCAT or other Visual Studio projects.



.NET Framework 4.5.2 Sort by: Default

Type: TwinCAT Connectivity Project

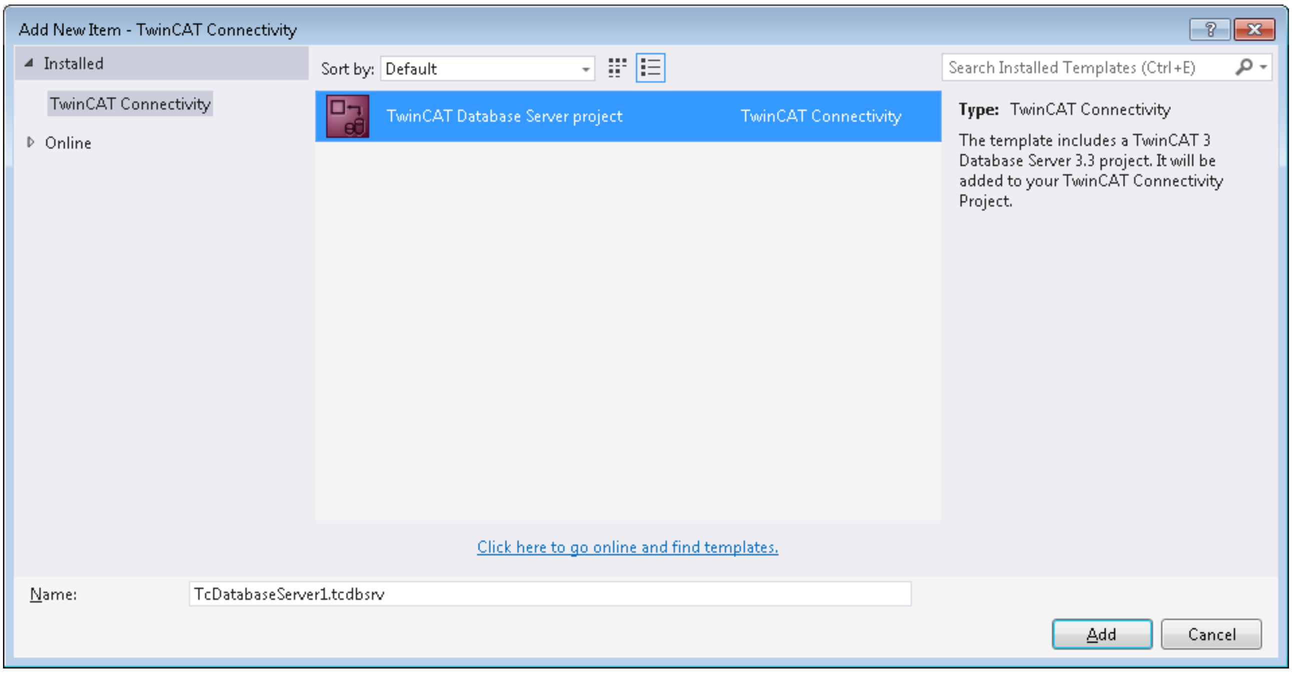
A template for creating an empty TwinCAT Connectivity project. You can add different kinds of TwinCAT Connectivity Functions to this project.

Name: TwinCAT Connectivity1

Location: C:\Users\ManuelA.BECKHOFF\Documents\Visual Studio 2015\Projects\TwinCAT Connectivity

A new project node appears in the solution. Below the Connectivity project node you can add subprojects for the supported connectivity functions.

Use **Add** to add a new TwinCAT Database Server project to the TwinCAT Connectivity project. The TwinCAT Database Server project can be found in the list of existing Item Templates.



Add New Item - TwinCAT Connectivity

Installed
TwinCAT Connectivity
Online
Sort by: Default
TwinCAT Database Server project
TwinCAT Connectivity
Search Installed Templates (Ctrl+E)

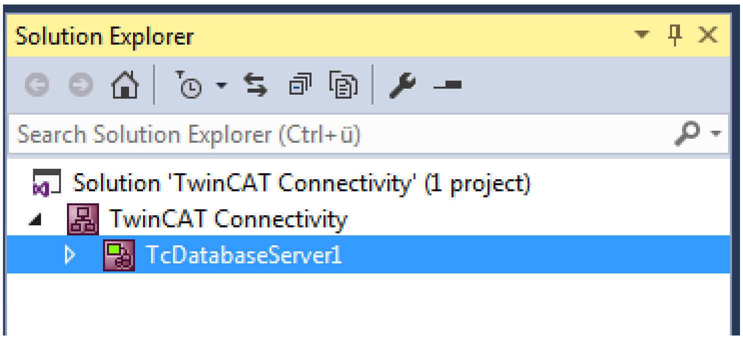
Type: TwinCAT Connectivity

The template includes a TwinCAT 3 Database Server 3.3 project. It will be added to your TwinCAT Connectivity Project.

Click here to go online and find templates.

Name: TcDatabaseServer1.tcdbsrv

A new TwinCAT Database Server project is created under the TwinCAT Connectivity node.

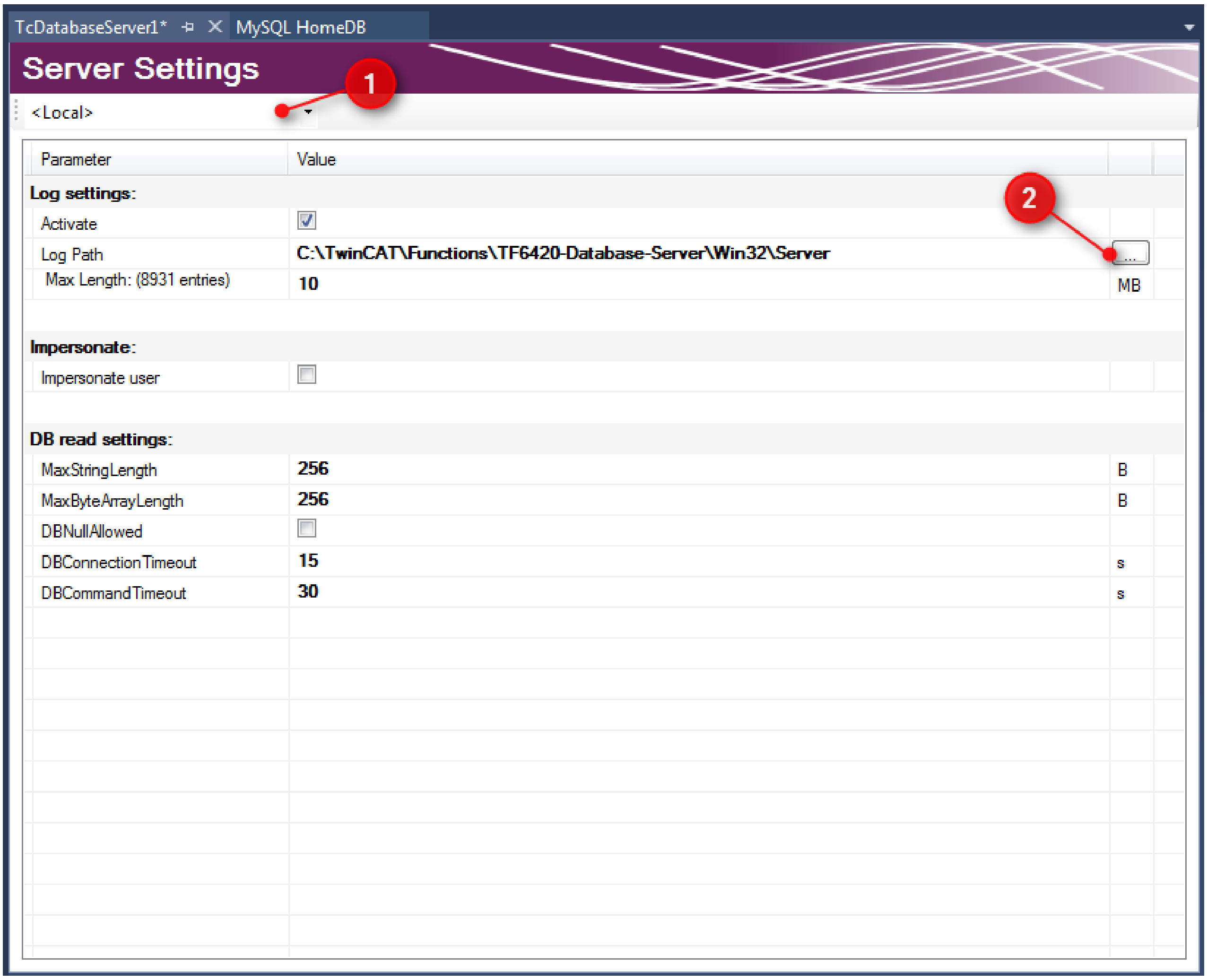


Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connectivity' (1 project)
TwinCAT Connectivity
TcDatabaseServer1

This is now used as the basis for the pending configuration of a TwinCAT Database Server. The document can be edited either via the Properties window or via an editor.

A Connectivity project can be associated with any number of TwinCAT Database Server projects or other projects, and it may therefore contain several configurations.

Editor for server settings



<Local>

Parameter	Value	
Log settings:		
Activate	☑	
Log Path	C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server	
Max Length: (8931 entries)	10	MB
Impersonate:		
Impersonate user	□	
DB read settings:		
MaxStringLength	256	B
MaxByteArrayLength	256	B
DBNullAllowed	□	
DBConnectionTimeout	15	s
DBCommandTimeout	30	s

The **Server Settings** editor can be used to edit the settings for the TwinCAT Database Server. These are general settings relating to the corresponding server. In the drop-down menu (1) you can select the target system via the Ams NetID. To this end you have to create a route to the target system via TwinCAT. When a finished configuration is transferred, the settings are stored in the TwinCAT Database Server for this target system.

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the Information Log Viewer [▶ 53]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.

For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

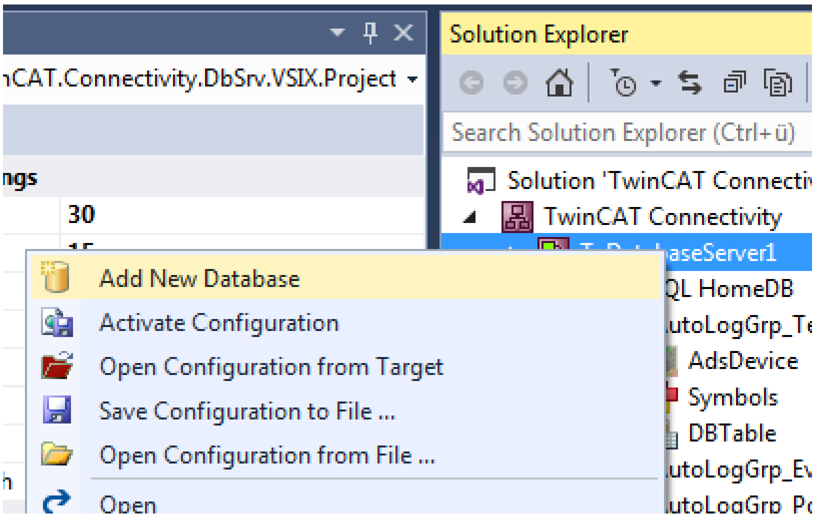
Further configuration settings are available to control the read process from the database. These settings refer to the TwinCAT Database Server on the target system:

MaxStringLength	Maximum string length of the variables in the PLC
MaxByteArrayLength	Maximum byte array length of the variables in the PLC
DBNullAllowed	Indicates whether ZERO values are accepted in the TwinCAT Database Server.
DBConnectionTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection error while attempts are made to establish a connection.
DBCommandTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection fault when a command was sent. If large data quantities are involved, processing of a command may take quite some time, depending on the database and the infrastructure.

Adding a new database configuration

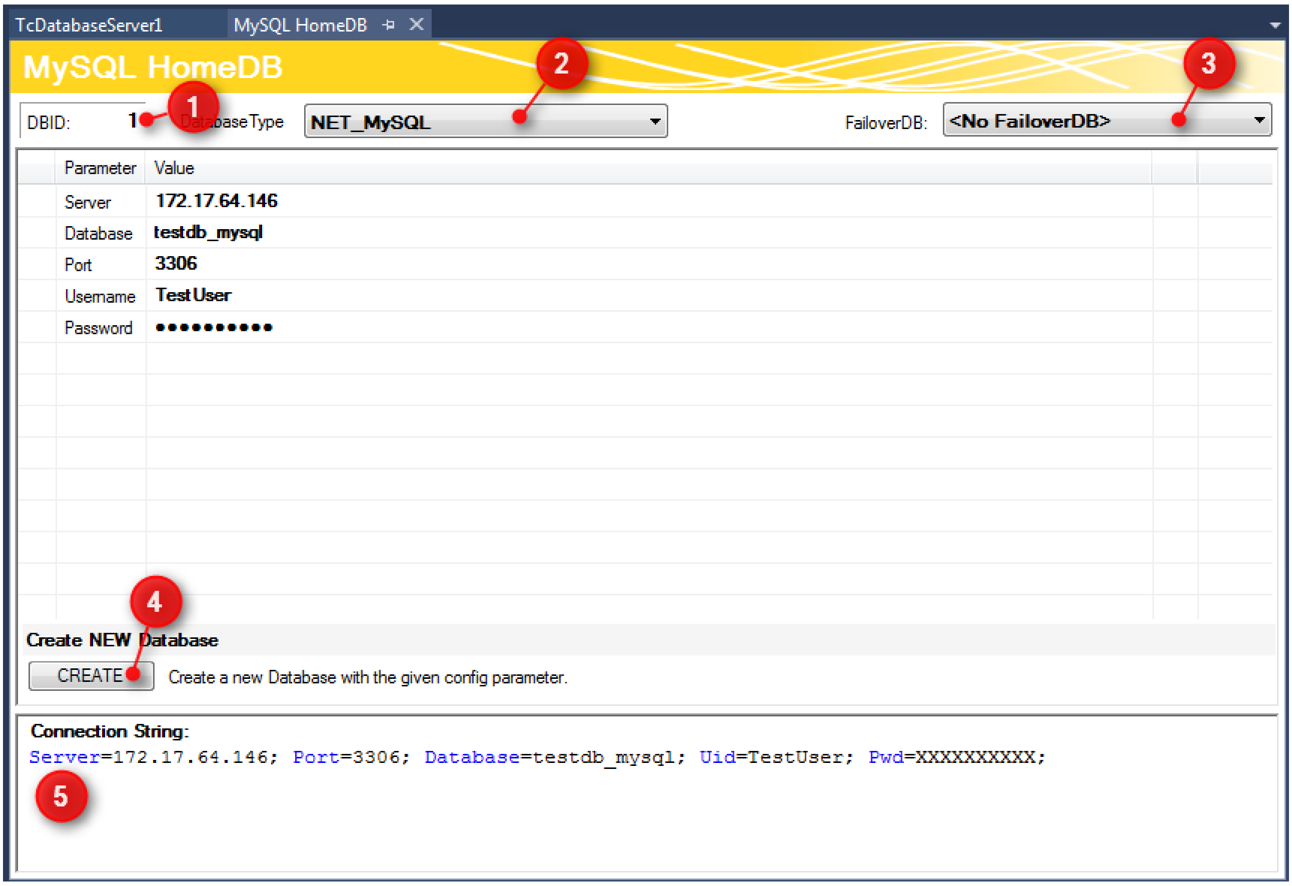
The database configuration is required for furnishing the Database Server with all the information required for the database connection.

A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.


ıCAT.Connectivity.DbSrv.VSIX.Project
ngs
30
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connecti
TwinCAT Connectivity
aseServer1
QL HomeDB
utoLogGrp_Te
AdsDevice
Symbols
DBTable
utoLogGrp_Ev
utoLogGrp_P
Add New Database
Activate Configuration
Open Configuration from Target
Save Configuration to File ...
Open Configuration from File ...
Open

A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

Editor for database configurations



DBID: 1 DatabaseType NET_MySQL FailoverDB: <No FailoverDB>

Parameter	Value
Server	172.17.64.146
Database	testdb_mysql
Port	3306
Username	TestUser
Password	●●●●●●●●●●●

Create NEW Database

CREATE Create a new Database with the given config parameter.

Connection String:

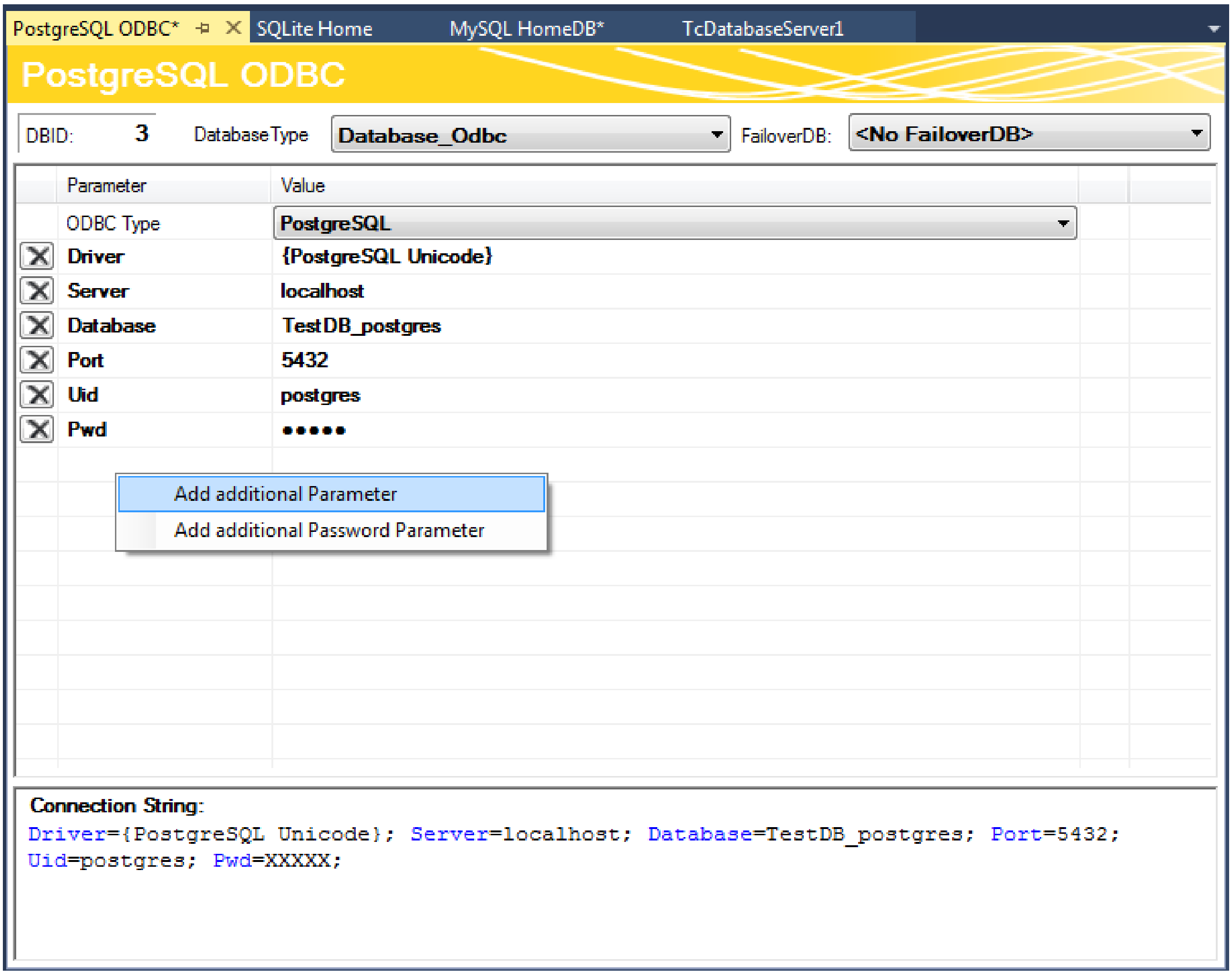
```
Server=172.17.64.146; Port=3306; Database=testdb_mysql; Uid=TestUser; Pwd=XXXXXXXXXX;
```

The database ID, which is required for some function blocks in the PLC, is shown in the upper part of the editor (1). The database types of the target database can be selected from the drop-down menu (2). Another option is the ODBC interface for a database, although this is not yet supported. Note that not all functions of the TwinCAT Database Server can be guaranteed, depending on the database.

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.

For each database [▶ 118] additional adjustable parameters are available. Depending on the database a connection string (5) is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

The **CREATE** (4) button can be used to create a new database. This function is only displayed if the respective database supports it.



DBID: 3 DatabaseType Database_Odbc FailoverDB: <No FailoverDB>

	Parameter	Value
	ODBC Type	PostgreSQL
☒	Driver	{PostgreSQL Unicode}
☒	Server	localhost
☒	Database	TestDB_postgres
☒	Port	5432
☒	Uid	postgres
☒	Pwd	●●●●●

Connection String:

```
Driver={PostgreSQL Unicode}; Server=localhost; Database=TestDB_postgres; Port=5432;
Uid=postgres; Pwd=XXXXX;
```

Unknown databases can be configured via an ODBC interface. In the **ODBC Type** drop-down list select "Unknown Database" and add parameters via the commands in the context menu. They may contain passwords, which are stored in encrypted form. The required connection string can be assembled from these parameters. Note that only limited functions of the TwinCAT Database Server can be used. Only the explicit function blocks of the SQL Expert mode are supported.

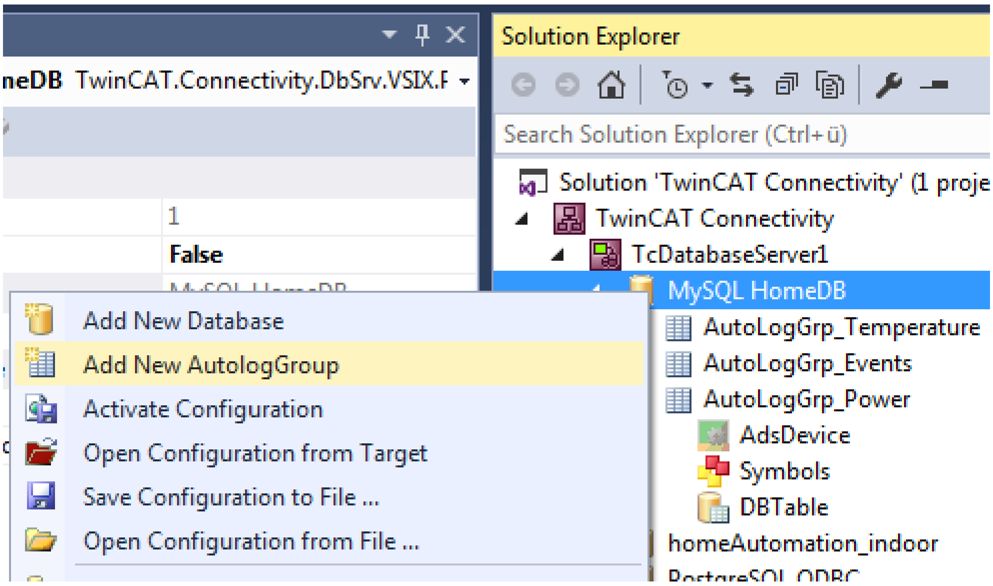
Failover database

The TwinCAT 3 Database Server has a failover database function. This function offers an option to switch to another database in the event of a connection loss or other problems with the database that was set up, in order to avoid possible data loss. This function is only supported by the Configure mode. In the case of automatic writing, the corresponding alternative database is used in the event of an error. The table of the first database must match the second.

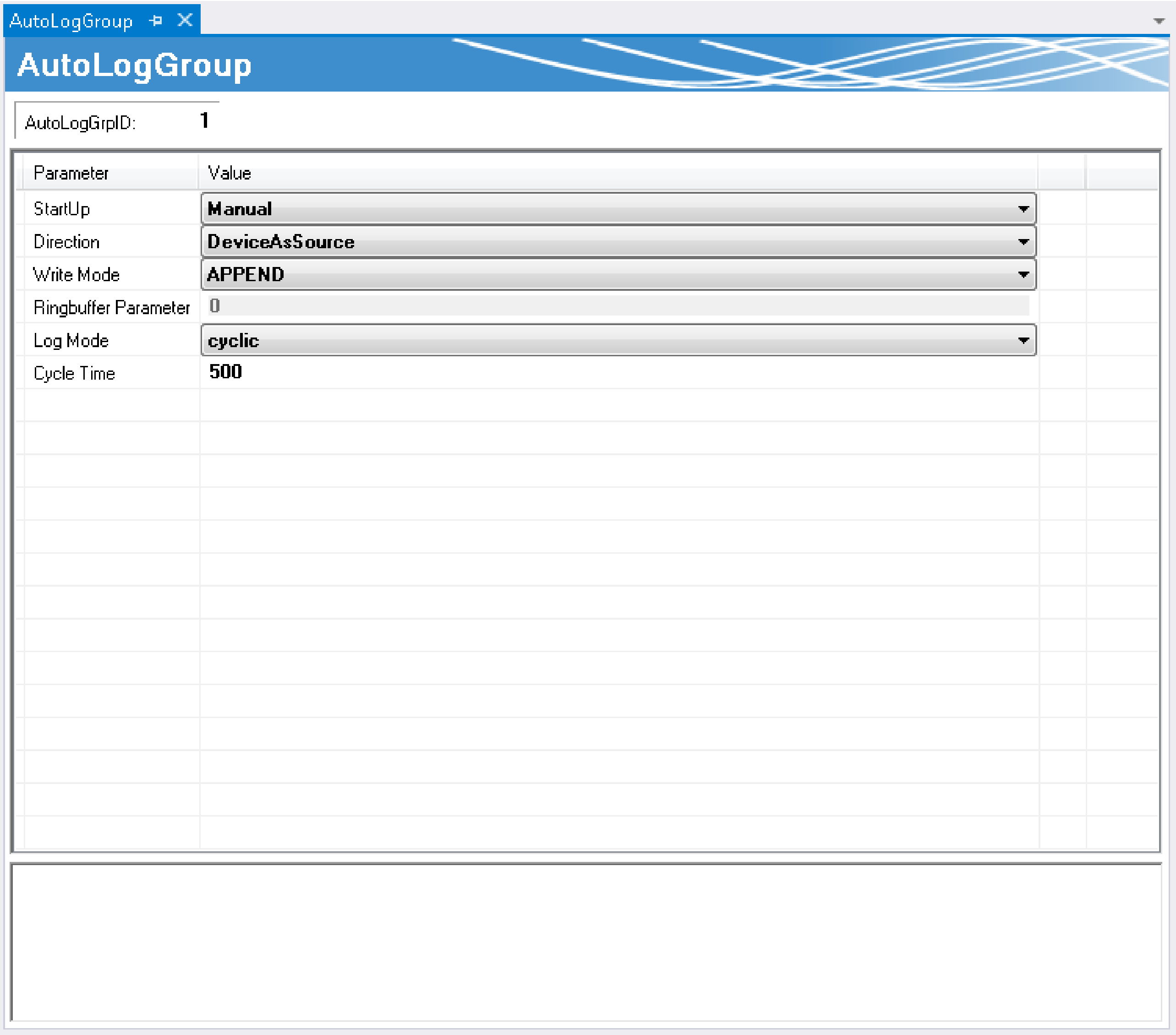
Adding a new AutoLog group

The AutoLog groups contain information on which variables of the PLC are to be synchronized with which variables from the databases. In addition, information about the synchronization times and the type of synchronization are stored here.

A new AutoLog group for the database configuration can be added via the command **Add New AutologGroup** in the context menu of a database configuration or via the toolbar. These AutoLog groups refer to the parent database.



A new AutoLog group and the corresponding components are added as files to the project folder and integrated in the project. They include the ADS device, the symbol groups and the table settings. In order to save these files in the project, you should save the TwinCAT Connectivity project file. The files can then be edited in editors or in the Properties window.



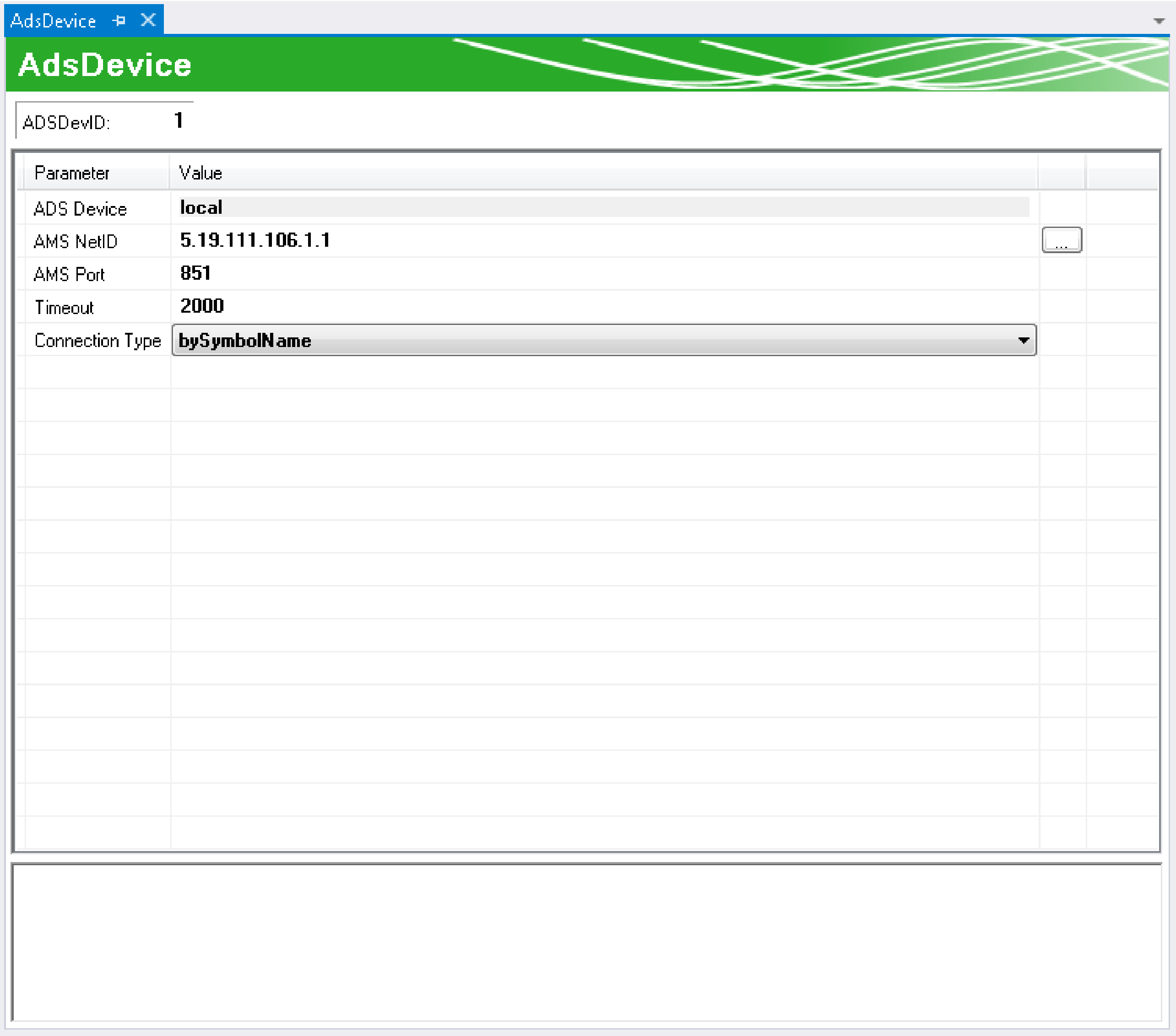
AutoLogGrpID: 1

Parameter	Value
StartUp	Manual
Direction	DeviceAsSource
Write Mode	APPEND
Ringbuffer Parameter	0
Log Mode	cyclic
Cycle Time	500

StartUp	AutoLog mode can be enabled manually (with a command in the PLC or from the configurator) or automatically during system startup.
Direction	The set ADS device is used as data target or data source.
Write mode	The data can appended in a database line-by-line, held in a ring buffer on a temporal or quantitative basis, or simply be updated at the corresponding position.
Ring buffer parameter	Depending on the setting this parameter represent the time or the cycles after which the ring buffer is updated.
Log mode	The variable is written either after a certain cycle time or when a change occurs.
Cycle Time	Cycle time after which the variable is written.

Configuring the ADS device

The ADS device is automatically created under an AutoLog group. In the most frequent use case the ADS device is the PLC runtime. The following parameters can be set in the editor:



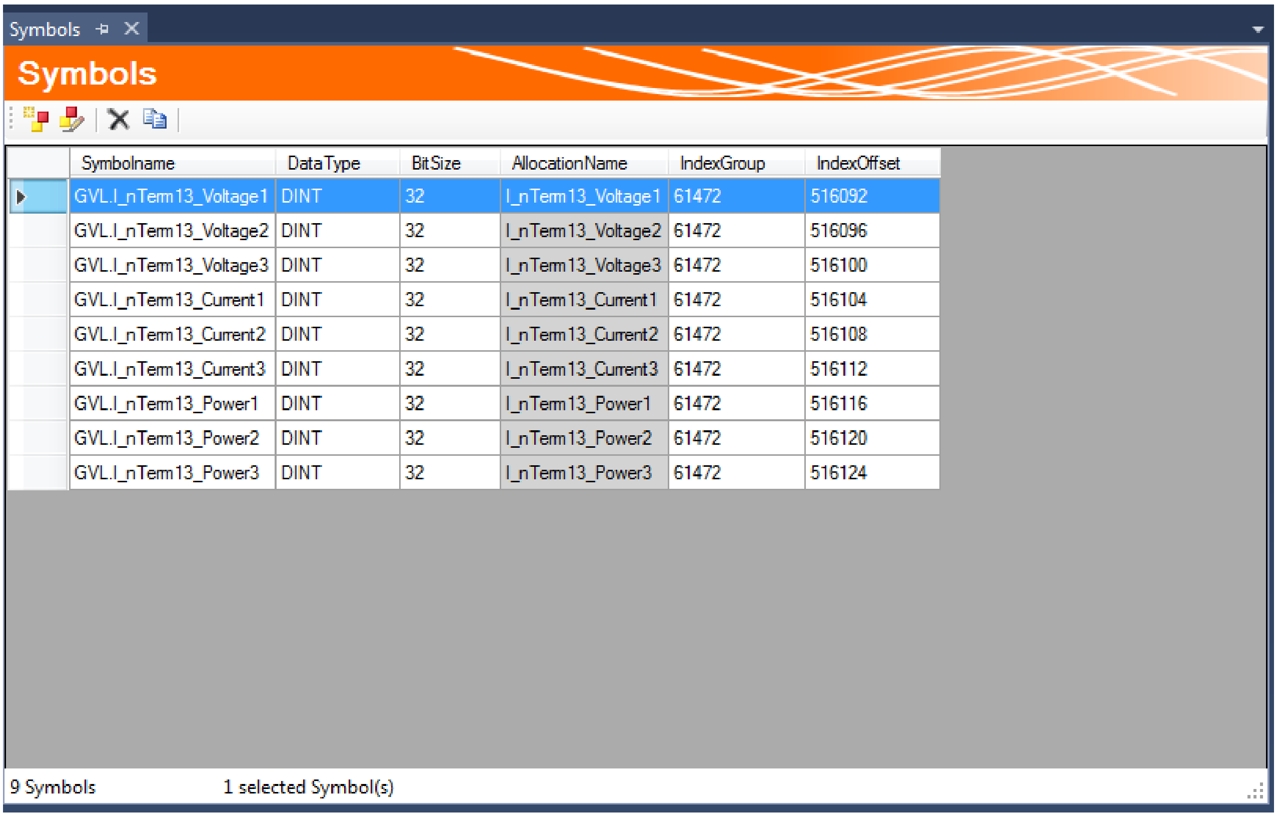
ADSDevID: 1

Parameter	Value
ADS Device	local
AMS NetID	5.19.111.106.1.1
AMS Port	851
Timeout	2000
Connection Type	bySymbolName

ADS Device	Name of the ADS target device.
AMS NetID	Address of the target device in the TwinCAT network.
AMS Port	Port of the target device in the TwinCAT network.
Timeout	Time after which it is assumed that the connection to the target device is lost.
Connection Type	bySymbolName: Connection is established based on the symbol name. byIndexGroup: Connection is established based on the memory index.

Configuring symbols

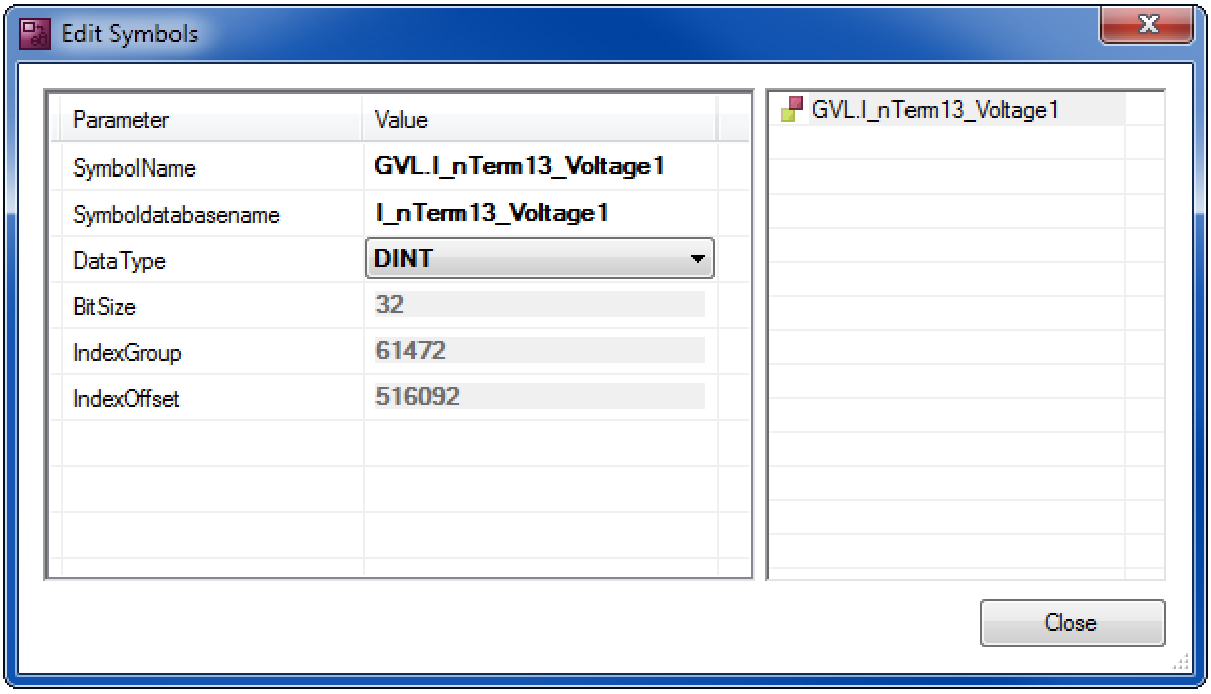
The symbols you set here are written to or read from the database, depending on whether the ADS device is the data target or the data source. The TwinCAT Target browser can be used for convenient access. Here you can search for the symbols on the target and communicate between the two tools via drag & drop.



Symbolname	DataType	BitSize	AllocationName	IndexGroup	IndexOffset
GVL.I_nTerm13_Voltage1	DINT	32	I_nTerm13_Voltage1	61472	516092
GVL.I_nTerm13_Voltage2	DINT	32	I_nTerm13_Voltage2	61472	516096
GVL.I_nTerm13_Voltage3	DINT	32	I_nTerm13_Voltage3	61472	516100
GVL.I_nTerm13_Current1	DINT	32	I_nTerm13_Current1	61472	516104
GVL.I_nTerm13_Current2	DINT	32	I_nTerm13_Current2	61472	516108
GVL.I_nTerm13_Current3	DINT	32	I_nTerm13_Current3	61472	516112
GVL.I_nTerm13_Power1	DINT	32	I_nTerm13_Power1	61472	516116
GVL.I_nTerm13_Power2	DINT	32	I_nTerm13_Power2	61472	516120
GVL.I_nTerm13_Power3	DINT	32	I_nTerm13_Power3	61472	516124

9 Symbols 1 selected Symbol(s)

Symbols can also be added manually to symbol groups or edited. The information that is required varies, depending on whether in the ADS device the connection type was selected via the symbol name or the index groups. The starting point is always the ADS device.



Edit Symbols

Parameter	Value
SymbolName	GVL.I_nTerm13_Voltage1
Symboldatabasename	I_nTerm13_Voltage1
DataType	DINT
BitSize	32
IndexGroup	61472
IndexOffset	516092

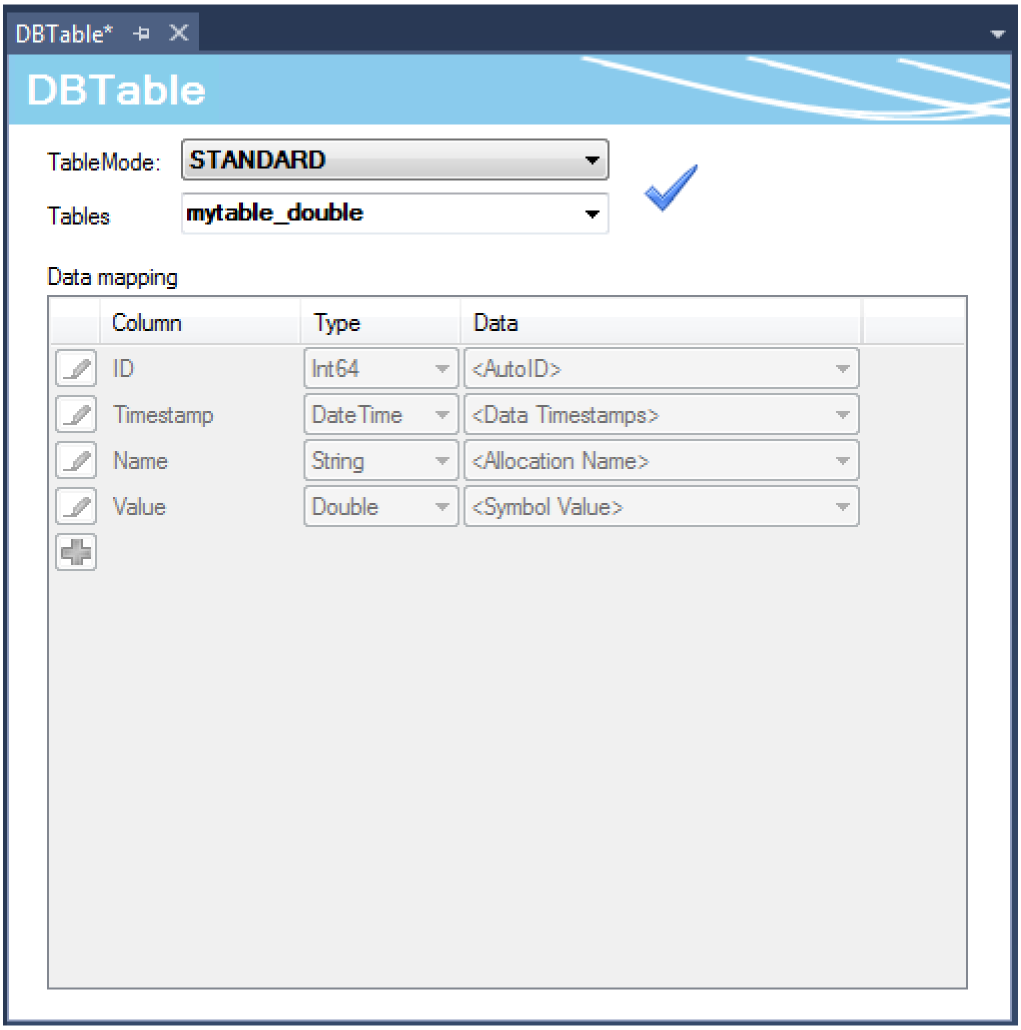
GVL.I_nTerm13_Voltage1

SymbolName	The symbol is addressed based on the set ADS device
Symbol database name	Name of the variable in the database table
DataType	PLC data type of the symbol
BitSize	Bit size of the symbols (set automatically for the data types)
IndexGroup	Index group in the TwinCAT system
IndexOffset	Index offset in the TwinCAT system

Configuring a table

The table in a database can be based on a standard table structure or on an individual structure.

The corresponding table can be selected from a list of possible tables. If the table does not yet exist, you can create it via the SQL Query Editor [▶ 43]. If you select the standard table structure, a blue tick indicates whether the selected table corresponds to this structure.



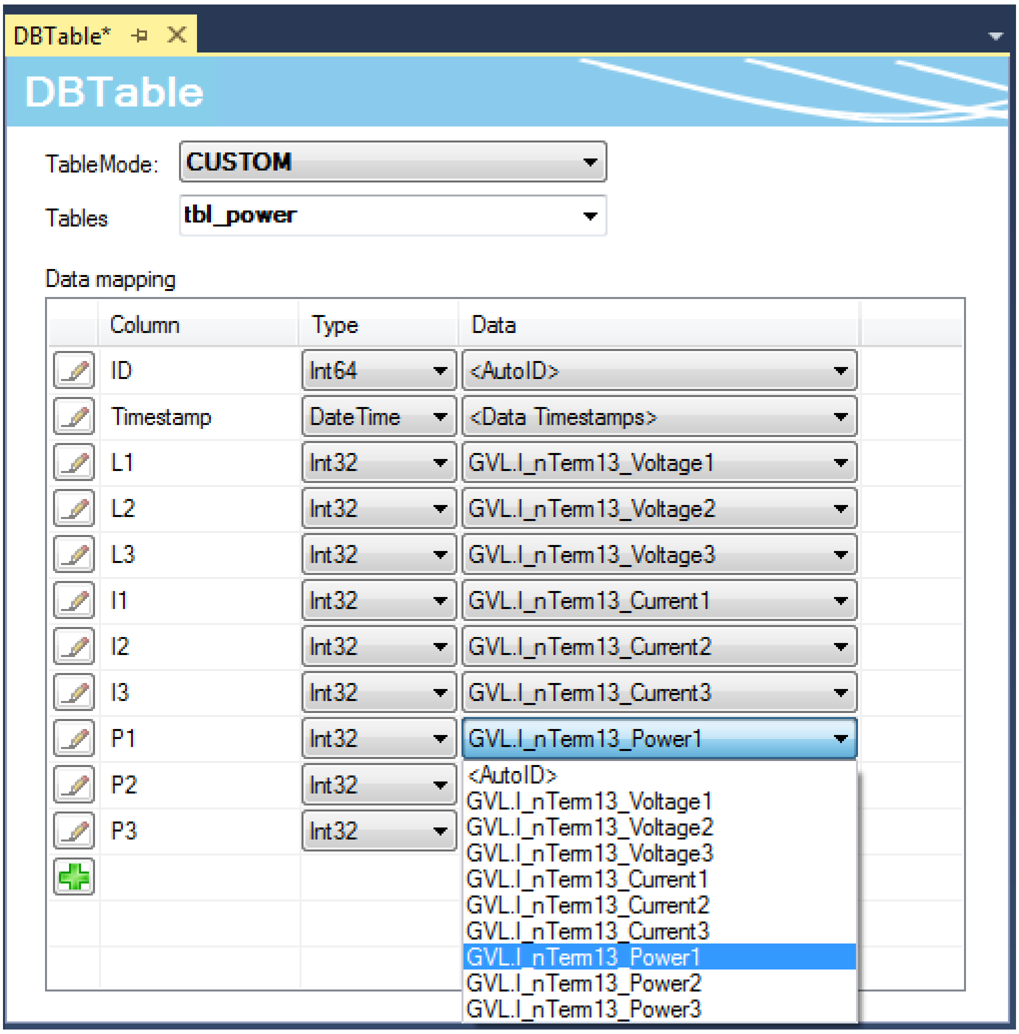
TableMode: STANDARD

Tables mytable_double

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
Name	String	<Allocation Name>
Value	Double	<Symbol Value>

The specific table type offers the option to distribute the individual symbols that were set in the symbol group to the table columns in the database as required. When a record is written to the database in AutoLog mode, the current values of the symbol group at the sampling time are saved in the corresponding table column.



DBTable

TableMode: CUSTOM

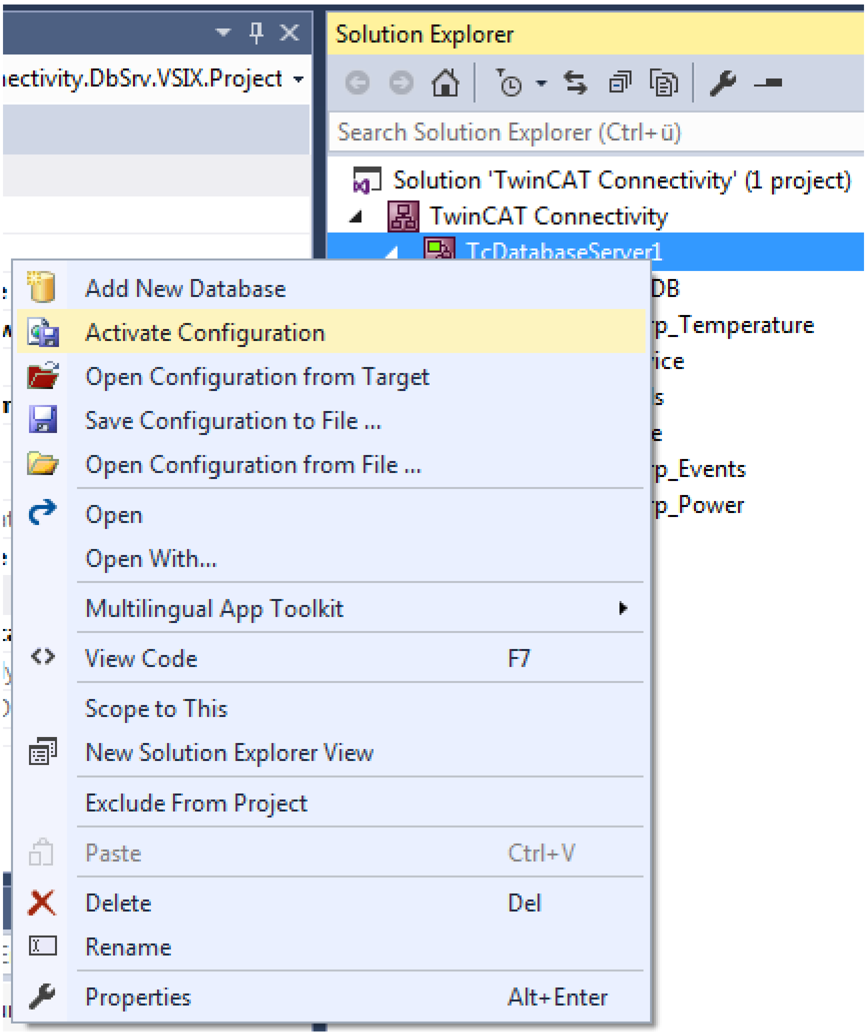
Tables tbl_power

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
L1	Int32	GVL.I_nTerm13_Voltage1
L2	Int32	GVL.I_nTerm13_Voltage2
L3	Int32	GVL.I_nTerm13_Voltage3
I1	Int32	GVL.I_nTerm13_Current1
I2	Int32	GVL.I_nTerm13_Current2
I3	Int32	GVL.I_nTerm13_Current3
P1	Int32	GVL.I_nTerm13_Power1
P2	Int32	
P3	Int32	

Activating a project

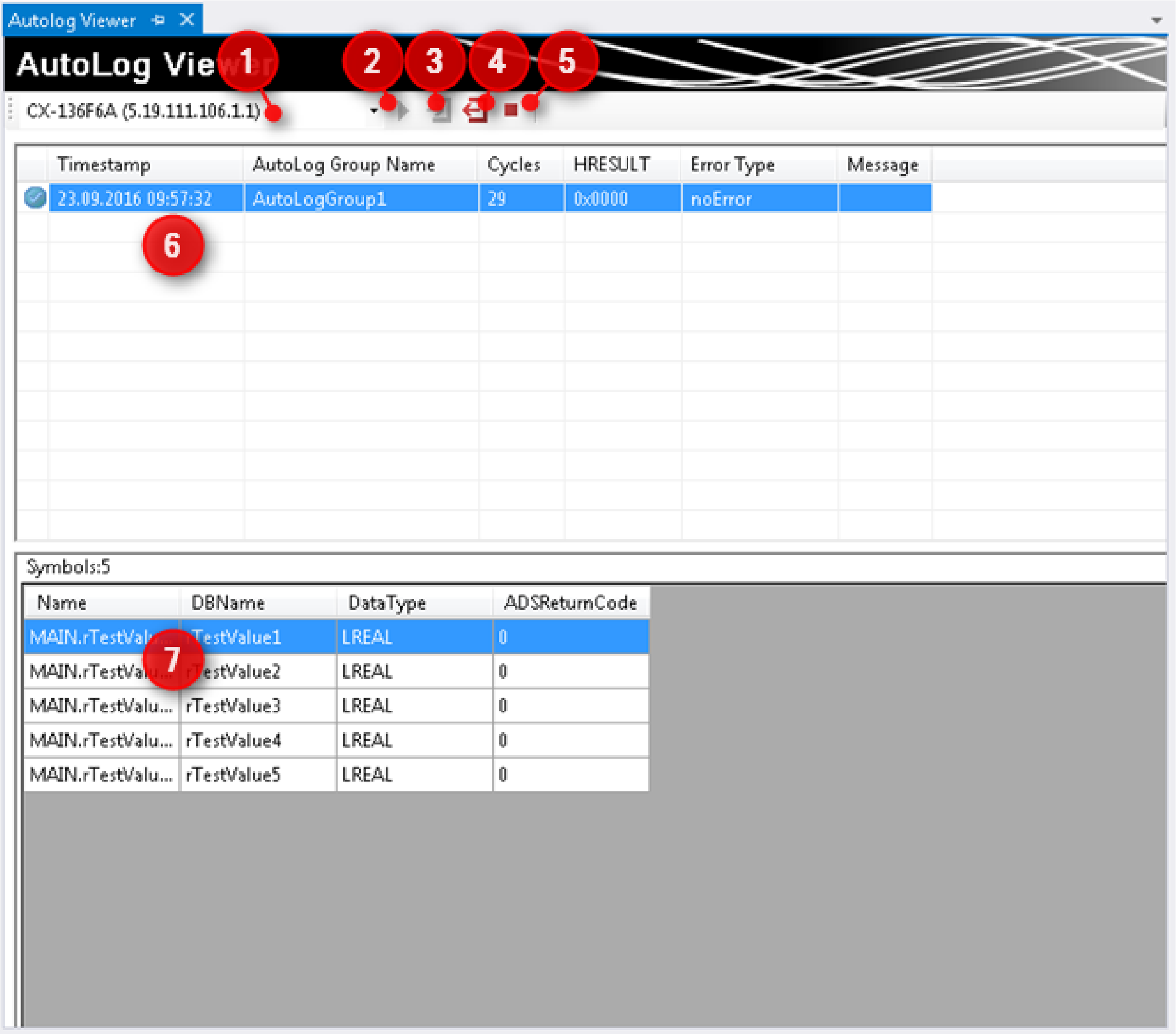
To activate a configured project on the TwinCAT Database Server, use the command **Activate Configuration** in the context menu of the TwinCAT Database Server project.



Logging of the variable starts when the TwinCAT system starts, depending on which startup behavior was specified in the AutoLog group. The mode can be started manually via the following AutoLog Viewer or with the corresponding function block from the PLC.

AutoLog Viewer

The AutoLog Viewer of the TwinCAT Database Server is a tool for controlling and monitoring the AutoLog mode. You can log into a target system, similar to the TwinCAT PLC. In logged-in state the AutoLog mode can be started or stopped. Information on the current state of the logging is shown in the lower part of the window. When an AutoLog group is selected, further information is displayed via the logged symbols.



CX-136F6A (5.19.111.106.1.1)

Timestamp	AutoLog Group Name	Cycles	HRESULT	Error Type	Message
23.09.2016 09:57:32	AutoLogGroup1	29	0x0000	noError	

Symbols:5

Name	DBName	DataType	ADSReturnCode
MAIN.rTestVal...	rTestValue1	LREAL	0
MAIN.rTestValu...	rTestValue2	LREAL	0
MAIN.rTestValu...	rTestValue3	LREAL	0
MAIN.rTestValu...	rTestValue4	LREAL	0
MAIN.rTestValu...	rTestValue5	LREAL	0

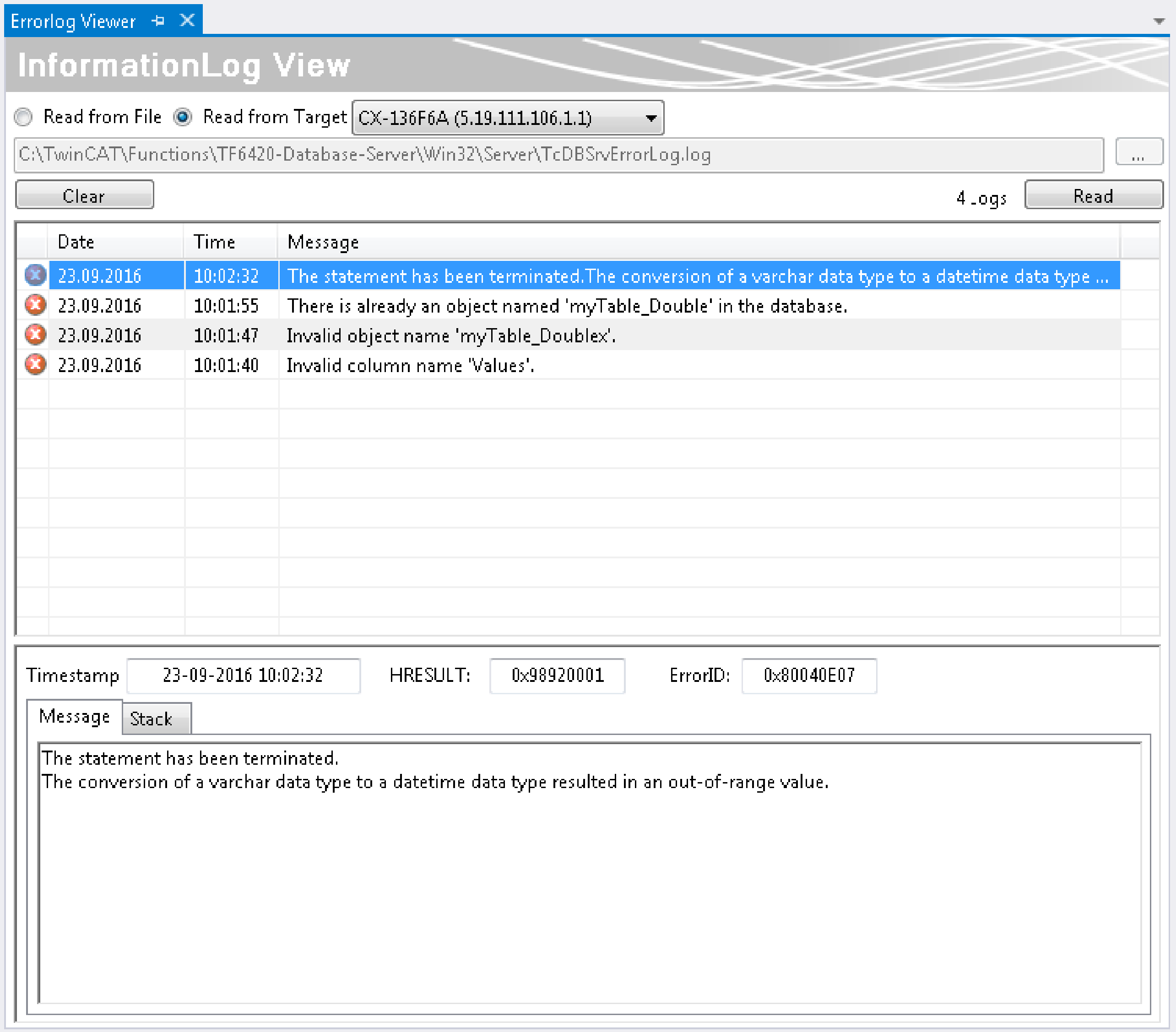
ID	Name	Function
1	Target system	Choose Target System with installed TwinCAT Database Server
2	Start	Manual start of the AutoLog mode
3	Login	Logging into the active AutoLog process
4	Logout	Logging out of the active AutoLog process
5	Stop	Manual stop of the AutoLog mode
6	AutoLog groups	List of configured AutoLog groups on the target system
7	Symbols	List of configured symbols for the selected AutoLog group

The AutoLog Viewer can be used to start and monitor the configured application. Depending on setting, after login and startup the incrementing cycle counter of the AutoLog group is visible according to the update times. Update errors are also shown here. For more detailed handling we recommend the InformationLog View.

More detailed error handling with the InformationLog View

InformationLog View is a tool for reading log files from the TwinCAT Database Server. Recorded information is displayed with a timestamp, IDs and error messages in plain text.

The log files can not only be viewed or emptied via direct file access, but also directly via the target. This is particularly advantageous with distributed Database Servers in a network, for quick and easy access to the log file. For this access a route to the target device must exist.



Errorlog Viewer
InformationLog View
○ Read from File ● Read from Target CX-136F6A (5.19.111.106.1.1)
C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server\TcDBSrvErrorLog.log
...
Clear
4 Logs
Read

Date	Time	Message
23.09.2016	10:02:32	The statement has been terminated.The conversion of a varchar data type to a datetime data type ...
23.09.2016	10:01:55	There is already an object named 'myTable_Double' in the database.
23.09.2016	10:01:47	Invalid object name 'myTable_Doublex'.
23.09.2016	10:01:40	Invalid column name 'Values'.

Timestamp: 23-09-2016 10:02:32 HRESULT: 0x98920001 ErrorID: 0x80040E07

```
The statement has been terminated.
The conversion of a varchar data type to a datetime data type resulted in an out-of-range value.
```

5.1.1.3 PLC Expert mode

This chapter is a compilation of all the information required for using the PLC Expert mode of the TwinCAT Database Server. In contrast to the Configure mode, in this mode data are not written or read based on cycles or events, but at specific times within the program sequence. This requires knowledge of the SQL language.

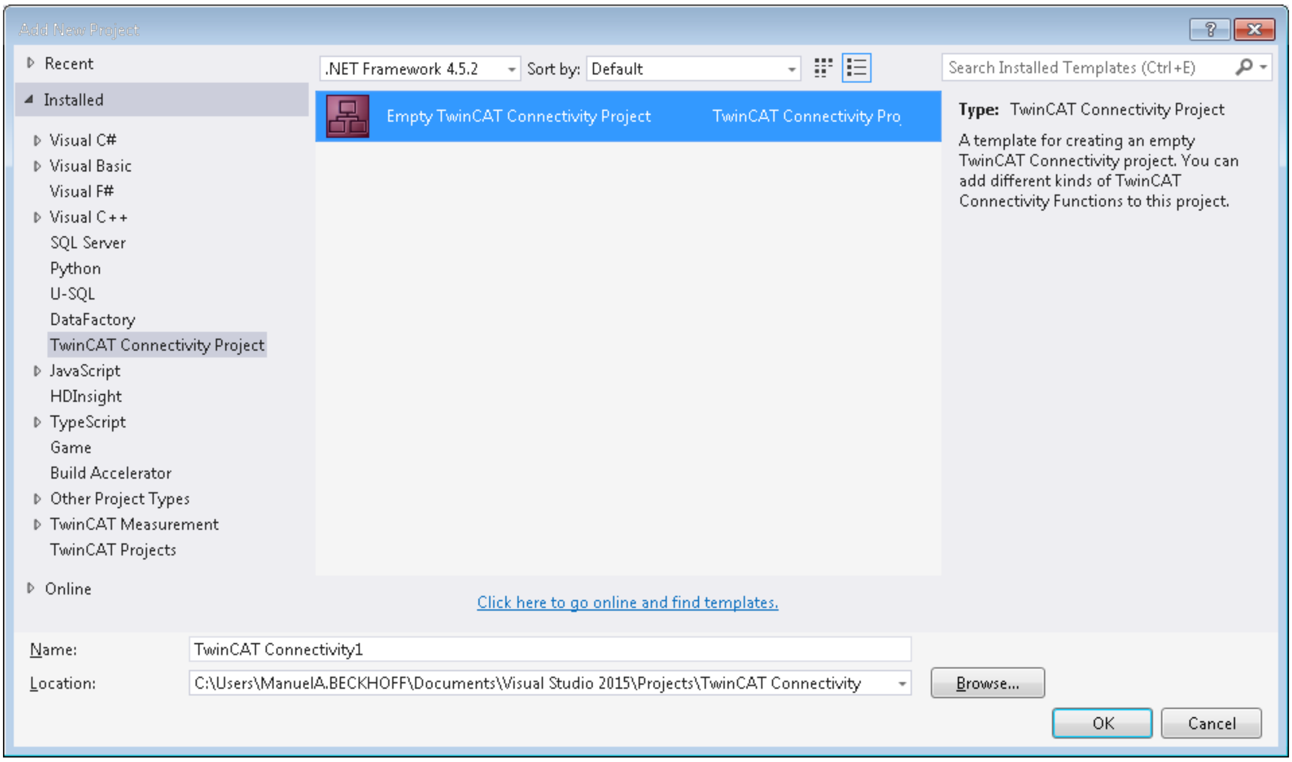
PLC Expert mode

In PLC Expert mode only the database configuration is set in the configurator. Further functionalities are implemented in the PLC code of the application. With the function block FB_PLCDBCreate [▶ 161] it is possible to dispense with the configurator and even configure the database itself from the PLC. Function blocks for reading and writing are available, if required. The function block FB_PLCDBCmd [▶ 173] forms the transition between PLC Expert mode and SQL Expert mode. Here, table structures can easily be mapped as PLC structures, and an SQL command with placeholders for the current structure values can be transferred to the TwinCAT Database Server. The TwinCAT Database Server then inserts all values automatically and sends the command to the database.

Build Project

The TwinCAT Connectivity extension for Visual Studio provides a new project template. When a new project is created, the **TwinCAT Connectivity Project** category appears as an option.

To create a new TwinCAT Connectivity project, select **Empty TwinCAT Connectivity Project**, specify the project name and the storage location and click **OK** to add it to the solution. In this way, TwinCAT Connectivity projects or TwinCAT Database Server projects can conveniently be created in parallel with TwinCAT or other Visual Studio projects.



.NET Framework 4.5.2 Sort by: Default

Type: TwinCAT Connectivity Project

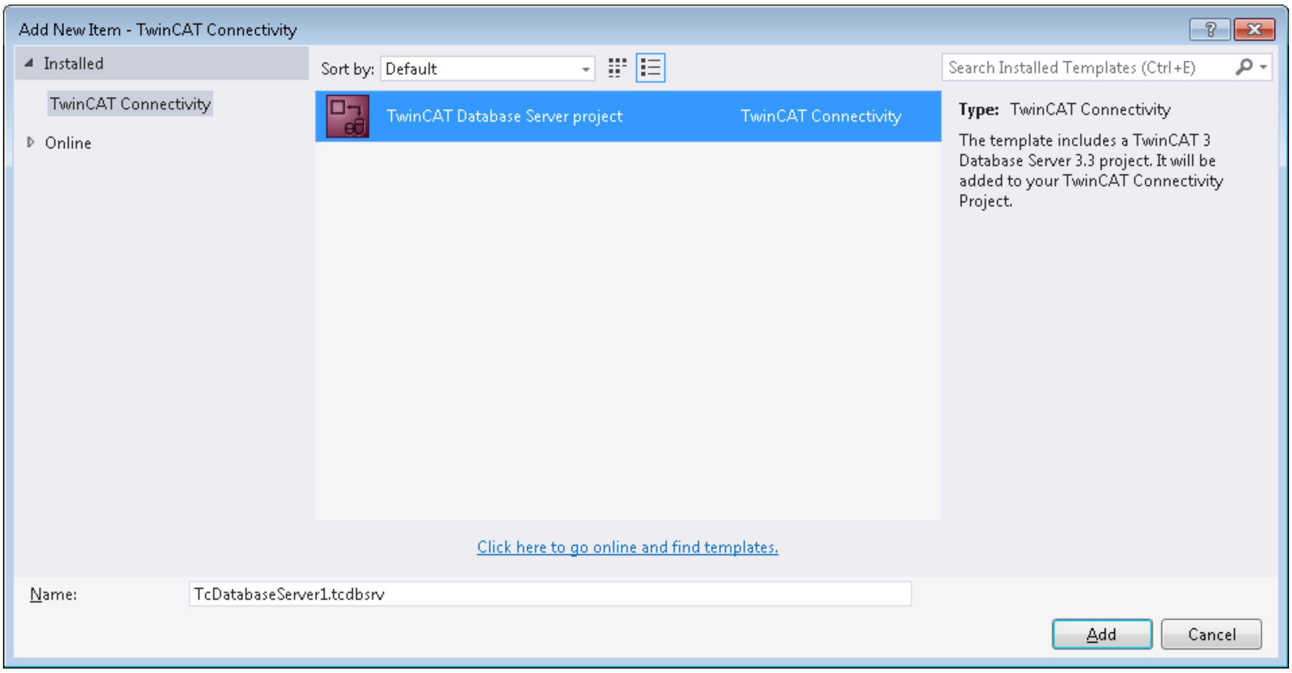
A template for creating an empty TwinCAT Connectivity project. You can add different kinds of TwinCAT Connectivity Functions to this project.

Name: TwinCAT Connectivity1

Location: C:\Users\ManuelA.BECKHOFF\Documents\Visual Studio 2015\Projects\TwinCAT Connectivity

A new project node appears in the solution. Below the Connectivity project node you can add subprojects for the supported connectivity functions.

Use **Add** to add a new TwinCAT Database Server project to the TwinCAT Connectivity project. The TwinCAT Database Server project can be found in the list of existing Item Templates.


Add New Item - TwinCAT Connectivity
Installed
TwinCAT Connectivity
Online
Search Installed Templates (Ctrl+E)
TwinCAT Database Server project
TwinCAT Connectivity
Click here to go online and find templates.
Add
Cancel

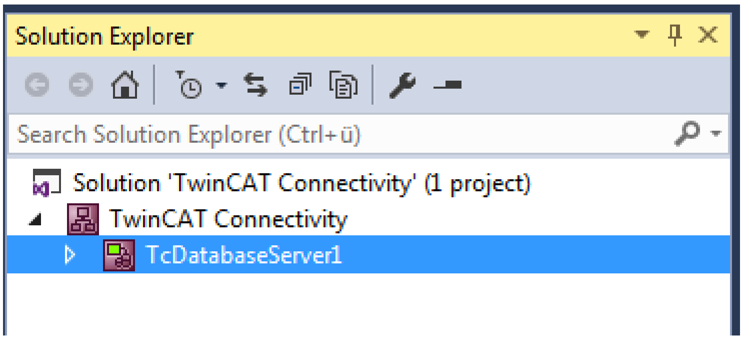
Sort by: Default

Type: TwinCAT Connectivity

The template includes a TwinCAT 3 Database Server 3.3 project. It will be added to your TwinCAT Connectivity Project.

Name: TcDatabaseServer1.tcdbsrv

A new TwinCAT Database Server project is created under the TwinCAT Connectivity node.

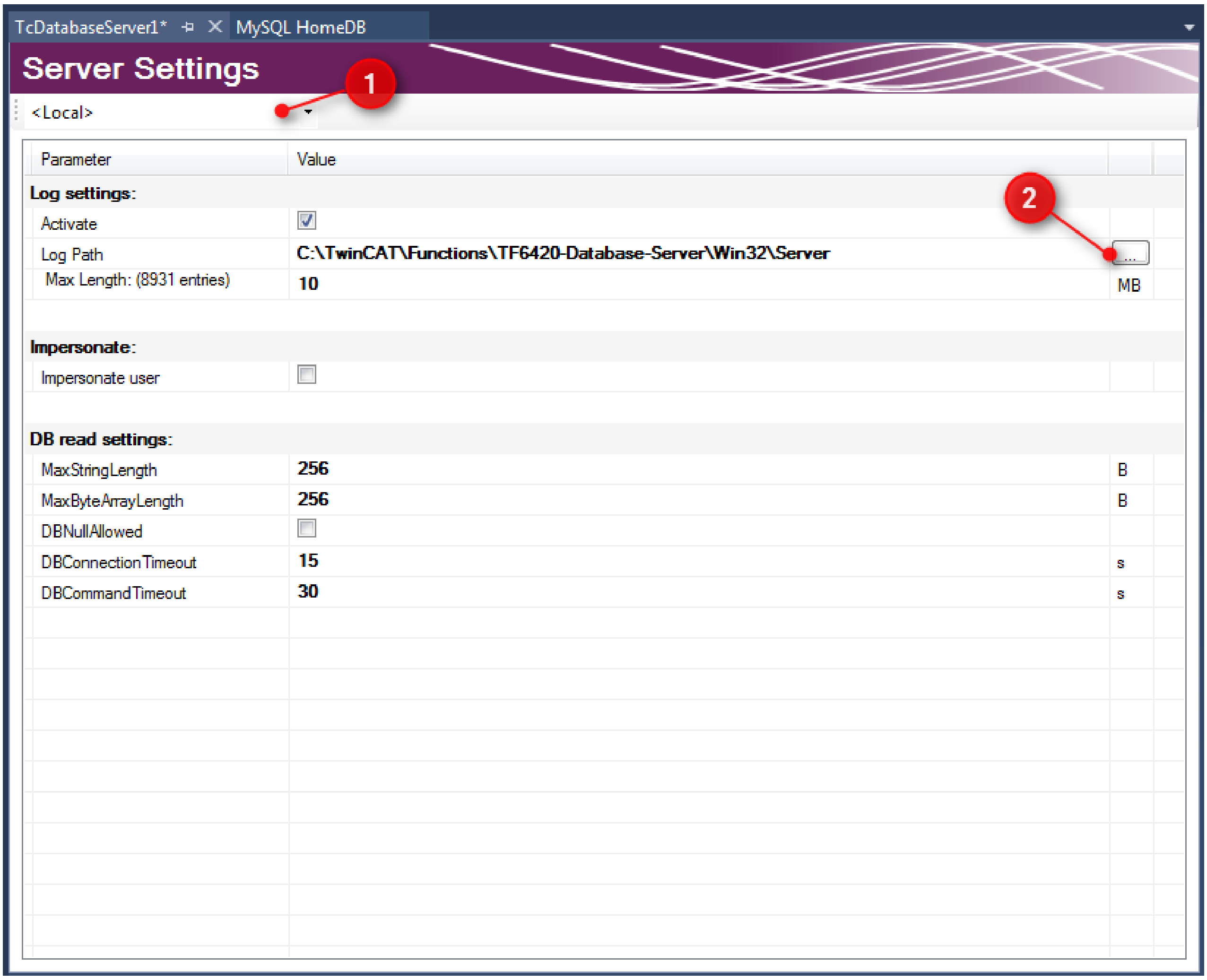


Solution 'TwinCAT Connectivity' (1 project)
TwinCAT Connectivity
TcDatabaseServer1

This is now used as the basis for the pending configuration of a TwinCAT Database Server. The document can be edited either via the Properties window or via an editor.

A Connectivity project can be associated with any number of TwinCAT Database Server projects or other projects, and it may therefore contain several configurations.

Editor for server settings



<Local>

Parameter	Value	
Log settings:		
Activate	☑	
Log Path	C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server	...
Max Length: (8931 entries)	10	MB
Impersonate:		
Impersonate user	☐	
DB read settings:		
MaxStringLength	256	B
MaxByteArrayLength	256	B
DBNullAllowed	☐	
DBConnectionTimeout	15	s
DBCommandTimeout	30	s

The **Server Settings** editor can be used to edit the settings for the TwinCAT Database Server. These are general settings relating to the corresponding server. In the drop-down menu (1) you can select the target system via the Ams NetID. To this end you have to create a route to the target system via TwinCAT. When a finished configuration is transferred, the settings are stored in the TwinCAT Database Server for this target system.

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the Information Log Viewer [▸ 53]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.

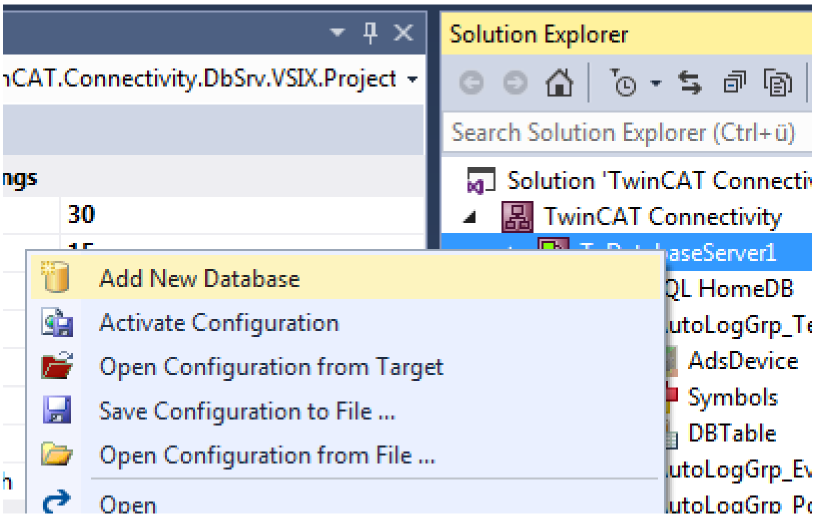
For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

Further configuration settings are available to control the read process from the database. These settings refer to the TwinCAT Database Server on the target system:

MaxStringLength	Maximum string length of the variables in the PLC
MaxByteArrayLength	Maximum byte array length of the variables in the PLC
DBNullAllowed	Indicates whether ZERO values are accepted in the TwinCAT Database Server.
DBConnectionTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection error while attempts are made to establish a connection.
DBCommandTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection fault when a command was sent. If large data quantities are involved, processing of a command may take quite some time, depending on the database and the infrastructure.

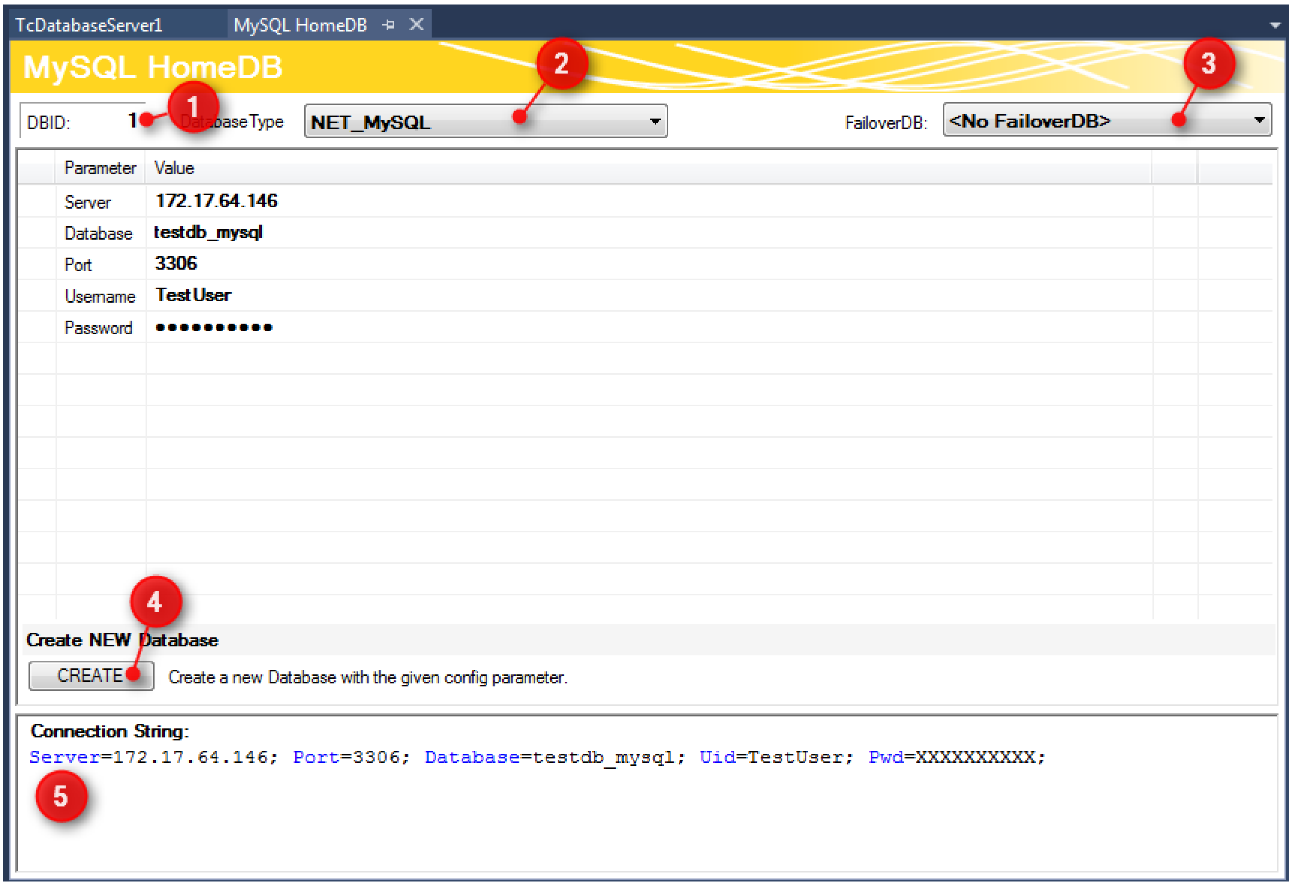
Adding a database configuration

A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.


ıCAT.Connectivity.DbSrv.VSIX.Project
ngs
30
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connecti
TwinCAT Connectivity
aseServer1
QL HomeDB
utoLogGrp_Te
AdsDevice
Symbols
DBTable
utoLogGrp_Ev
utoLogGrp_Pı
Add New Database
Activate Configuration
Open Configuration from Target
Save Configuration to File ...
Open Configuration from File ...
Open

A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

Editor for database configurations



DBID: 1 DatabaseType NET_MySQL FailoverDB: <No FailoverDB>

Parameter	Value
Server	172.17.64.146
Database	testdb_mysql
Port	3306
Username	TestUser
Password	●●●●●●●●●●●

Create NEW Database

CREATE Create a new Database with the given config parameter.

Connection String:

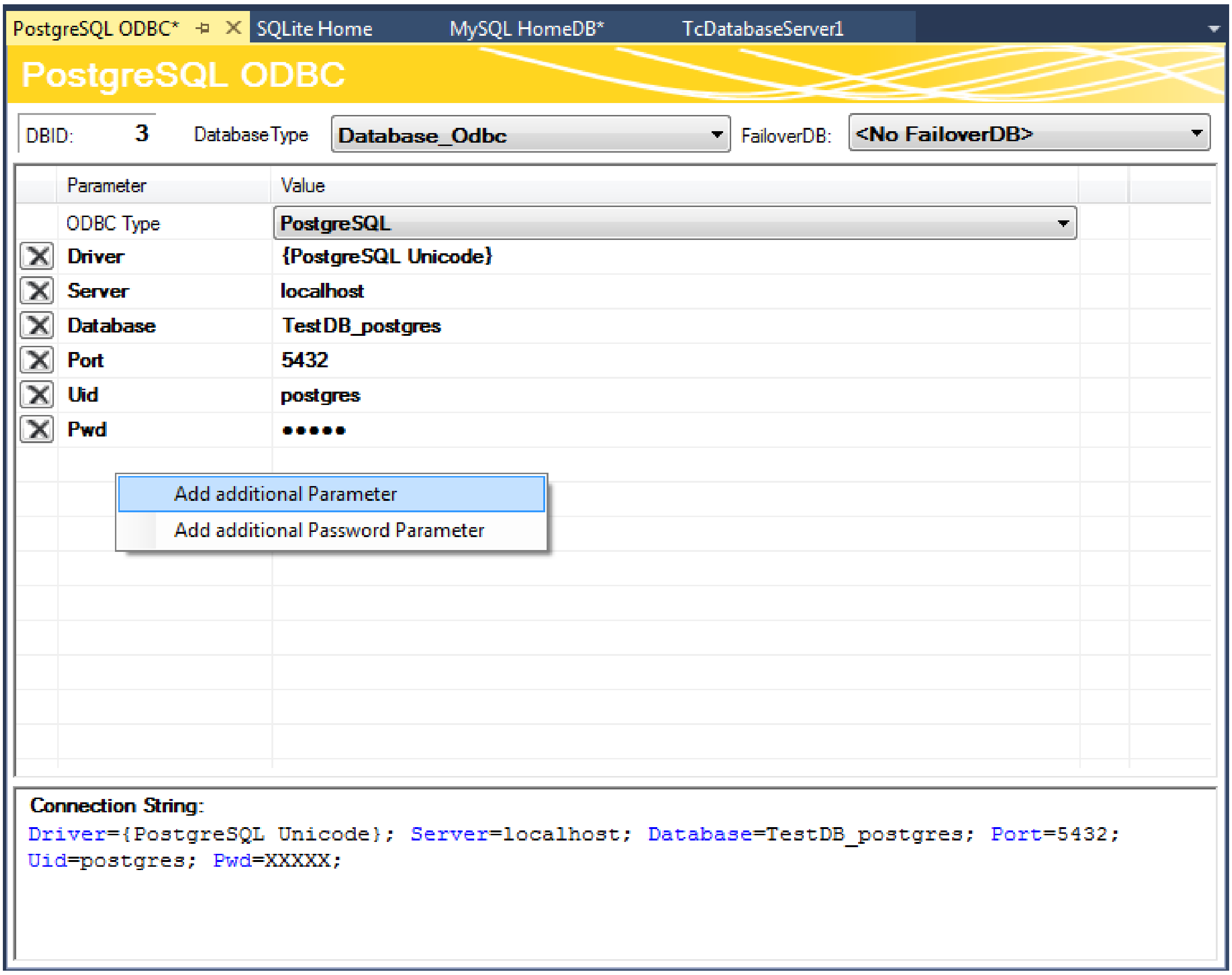
```
Server=172.17.64.146; Port=3306; Database=testdb_mysql; Uid=TestUser; Pwd=XXXXXXXXXX;
```

The database ID, which is required for some function blocks in the PLC, is shown in the upper part of the editor (1). The database types of the target database can be selected from the drop-down menu (2). Another option is the ODBC interface for a database, although this is not yet supported. Note that not all functions of the TwinCAT Database Server can be guaranteed, depending on the database.

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.

For each database [▶ 118] additional adjustable parameters are available. Depending on the database a connection string (5) is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

The **CREATE** (4) button can be used to create a new database. This function is only displayed if the respective database supports it.



DBID: 3 DatabaseType Database_Odbc FailoverDB: <No FailoverDB>

	Parameter	Value
	ODBC Type	PostgreSQL
☒	Driver	{PostgreSQL Unicode}
☒	Server	localhost
☒	Database	TestDB_postgres
☒	Port	5432
☒	Uid	postgres
☒	Pwd	●●●●●

Connection String:

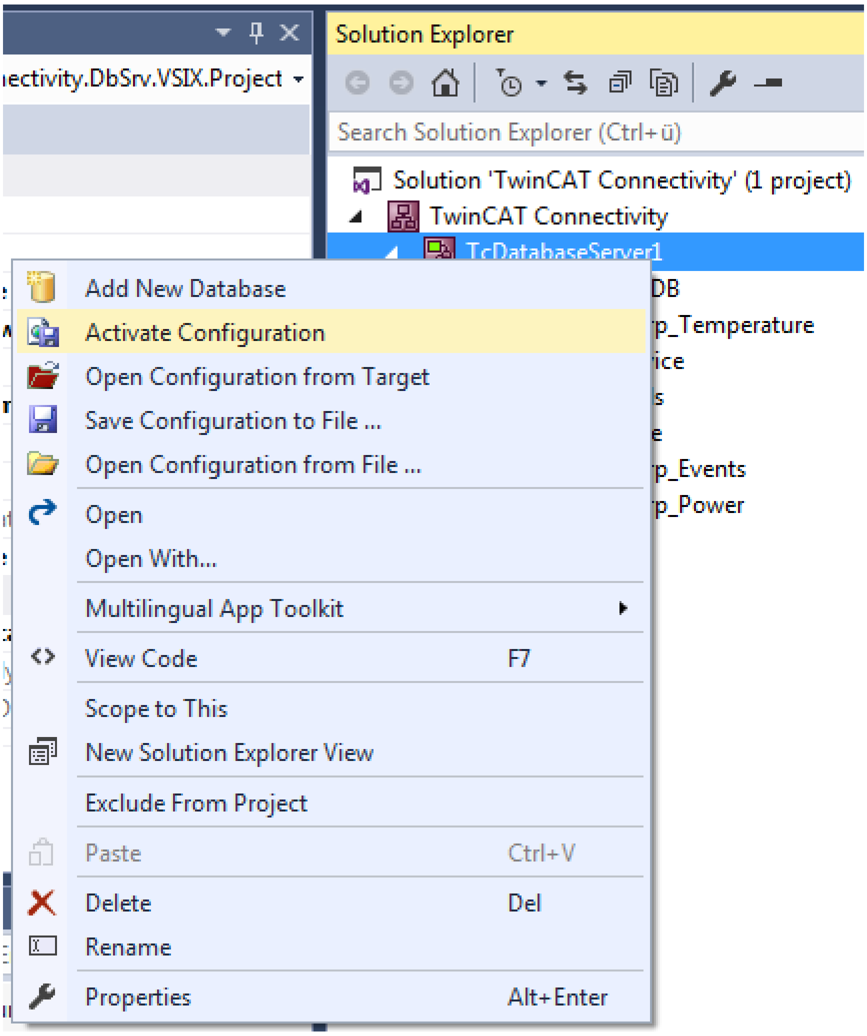
```
Driver={PostgreSQL Unicode}; Server=localhost; Database=TestDB_postgres; Port=5432;
Uid=postgres; Pwd=XXXXX;
```

Unknown databases can be configured via an ODBC interface. In the **ODBC Type** drop-down list select "Unknown Database" and add parameters via the commands in the context menu. They may contain passwords, which are stored in encrypted form. The required connection string can be assembled from these parameters. Note that only limited functions of the TwinCAT Database Server can be used. Only the explicit function blocks of the SQL Expert mode are supported.

No additional AutoLog group configuration is required in this mode, since writing and reading between the database and the PLC is called manually by the PLC programmer. The configuration part is now complete.

Activating a project

To activate a configured project on the TwinCAT Database Server, use the command **Activate Configuration** in the context menu of the TwinCAT Database Server project.



Once the project has been activated, the SQL Query Editor [▶ 43] can be used for further development steps, such as creating databases or tables, generating structures for the PLC, which match the corresponding table structure of the database, or testing connections to the database with the implemented information.

The PLC programmer can use the available PLC API [▶ 153] function blocks to communicate with the TwinCAT Database Server.

5.1.1.4 SQL Expert mode

This chapter describes all the steps required for using the SQL Expert mode. This mode is tailored for users with individual requirements. The following topics will be discussed:

1. Creating a project
2. Creating and setting up a database configuration
3. Activating a Database Server project
4. Creating SQL commands with the SQL Query Editor

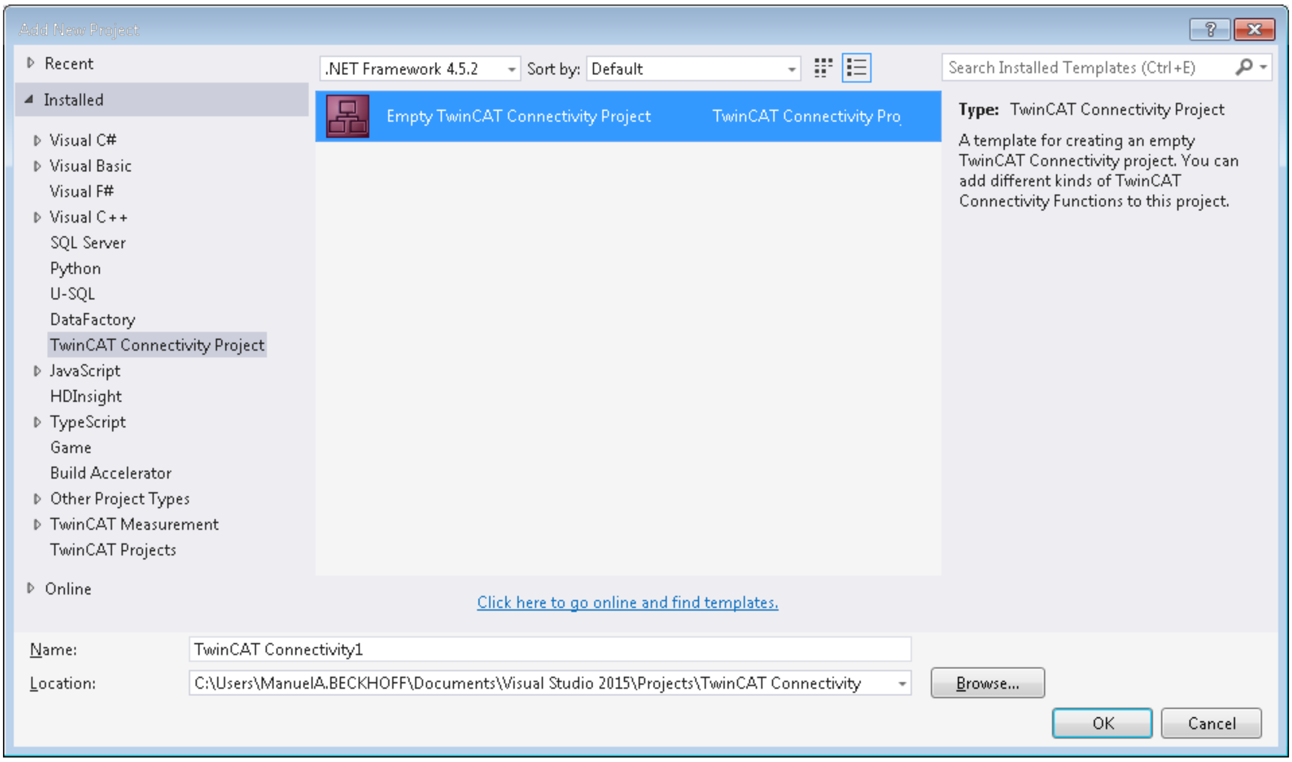
SQL Expert Mode

In SQL Expert mode users can assemble the SQL commands for Insert, Select or Update, for example, in the PLC and send them to the database via the TwinCAT Database Server. This is a very flexible and powerful option. Stored Procedures [▶ 191] - in database - can also be called from the PLC.

Build Project

The TwinCAT Connectivity extension for Visual Studio provides a new project template. When a new project is created, the **TwinCAT Connectivity Project** category appears as an option.

To create a new TwinCAT Connectivity project, select **Empty TwinCAT Connectivity Project**, specify the project name and the storage location and click **OK** to add it to the solution. In this way, TwinCAT Connectivity projects or TwinCAT Database Server projects can conveniently be created in parallel with TwinCAT or other Visual Studio projects.



Type: TwinCAT Connectivity Project

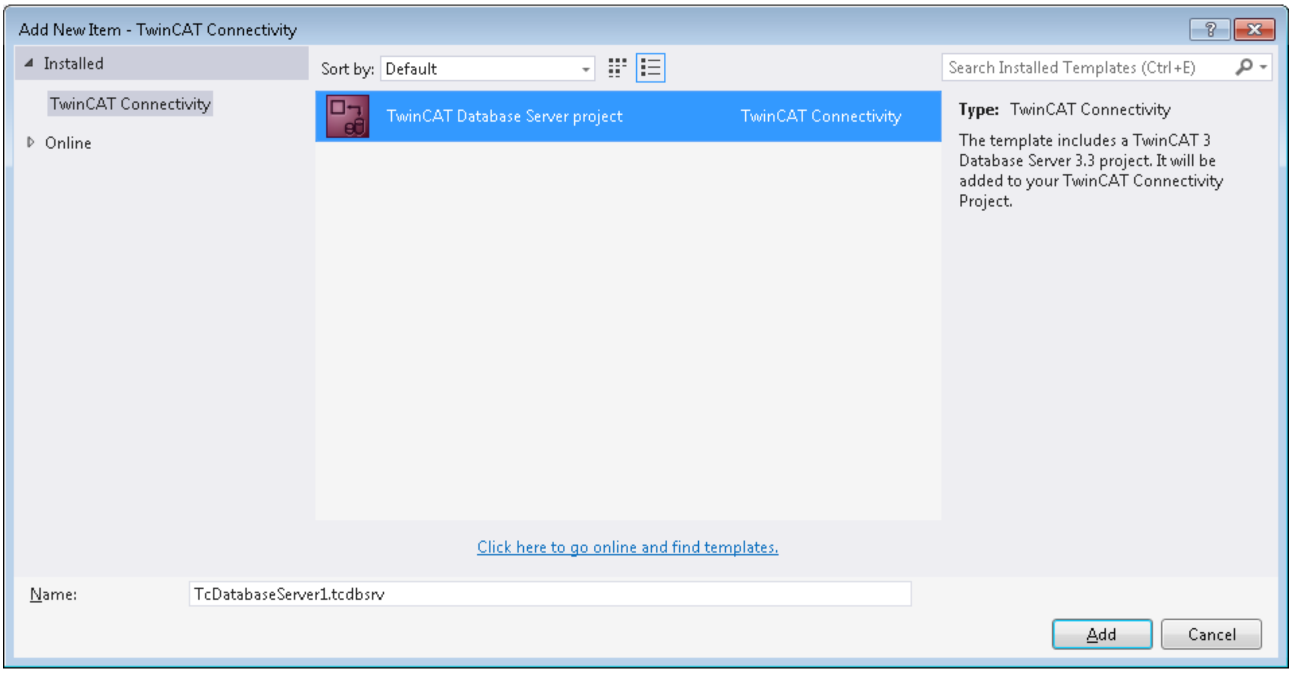
A template for creating an empty TwinCAT Connectivity project. You can add different kinds of TwinCAT Connectivity Functions to this project.

Name: TwinCAT Connectivity1

Location: C:\Users\ManuelA.BECKHOFF\Documents\Visual Studio 2015\Projects\TwinCAT Connectivity

A new project node appears in the solution. Below the Connectivity project node you can add subprojects for the supported connectivity functions.

Use **Add** to add a new TwinCAT Database Server project to the TwinCAT Connectivity project. The TwinCAT Database Server project can be found in the list of existing Item Templates.

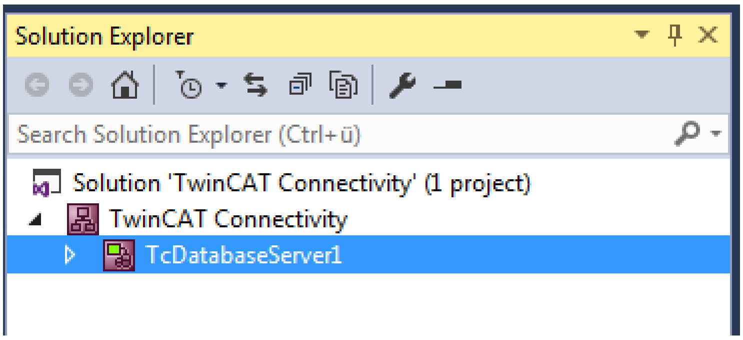

Add New Item - TwinCAT Connectivity
Installed
TwinCAT Connectivity
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
TwinCAT Database Server project
TwinCAT Connectivity
Click here to go online and find templates.
Add
Cancel

Type: TwinCAT Connectivity

The template includes a TwinCAT 3 Database Server 3.3 project. It will be added to your TwinCAT Connectivity Project.

Name: TcDatabaseServer1.tcdbsrv

A new TwinCAT Database Server project is created under the TwinCAT Connectivity node.

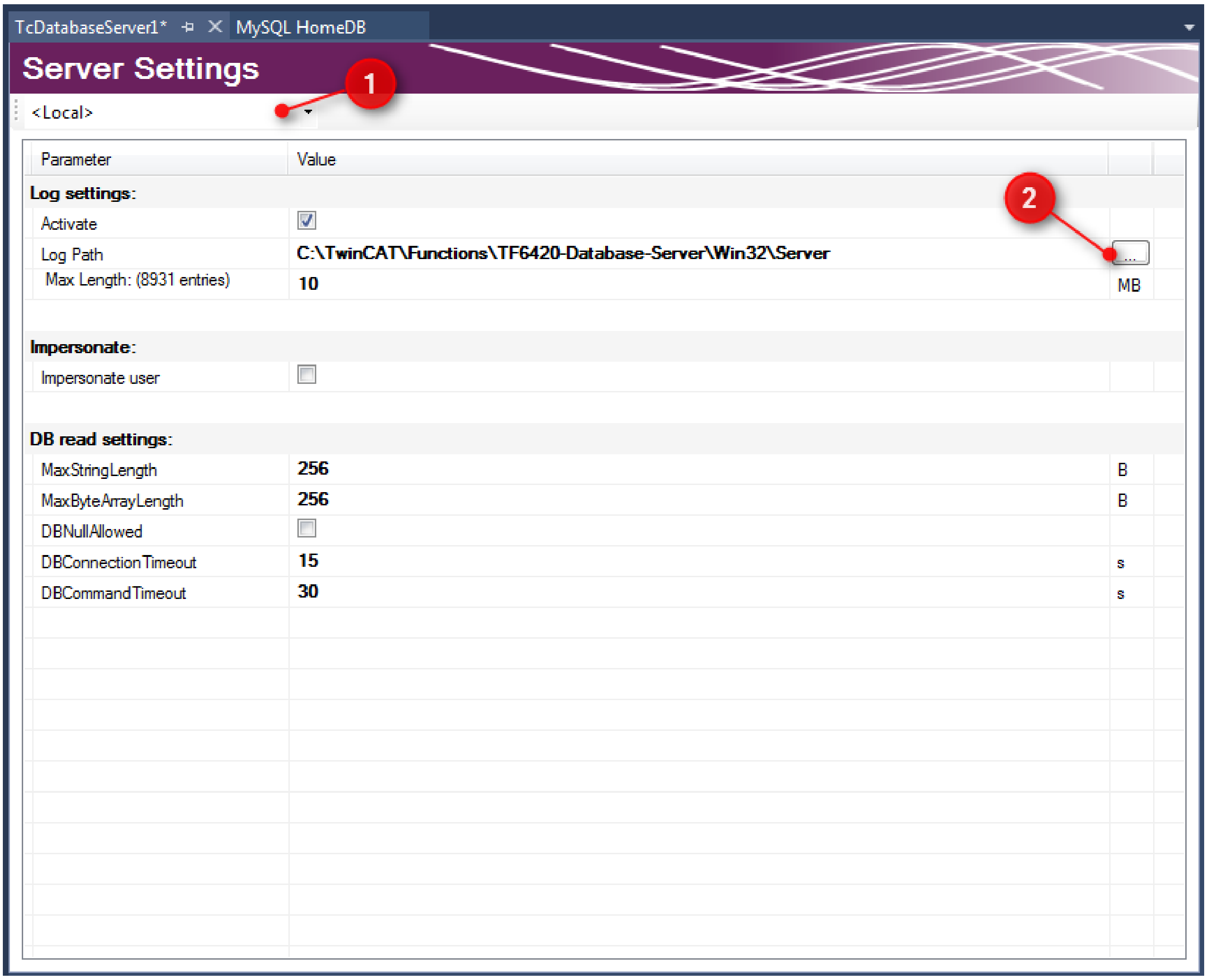


Solution 'TwinCAT Connectivity' (1 project)
- TwinCAT Connectivity
  - TcDatabaseServer1

This is now used as the basis for the pending configuration of a TwinCAT Database Server. The document can be edited either via the Properties window or via an editor.

A Connectivity project can be associated with any number of TwinCAT Database Server projects or other projects, and it may therefore contain several configurations.

Editor for server settings



<Local>

Parameter	Value	
Log settings:		
Activate	☑	
Log Path	C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server	
Max Length: (8931 entries)	10	MB
Impersonate:		
Impersonate user	☐	
DB read settings:		
MaxStringLength	256	B
MaxByteArrayLength	256	B
DBNullAllowed	☐	
DBConnectionTimeout	15	s
DBCommandTimeout	30	s

The **Server Settings** editor can be used to edit the settings for the TwinCAT Database Server. These are general settings relating to the corresponding server. In the drop-down menu (1) you can select the target system via the Ams NetID. To this end you have to create a route to the target system via TwinCAT. When a finished configuration is transferred, the settings are stored in the TwinCAT Database Server for this target system.

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the Information Log Viewer [▶ 53]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.

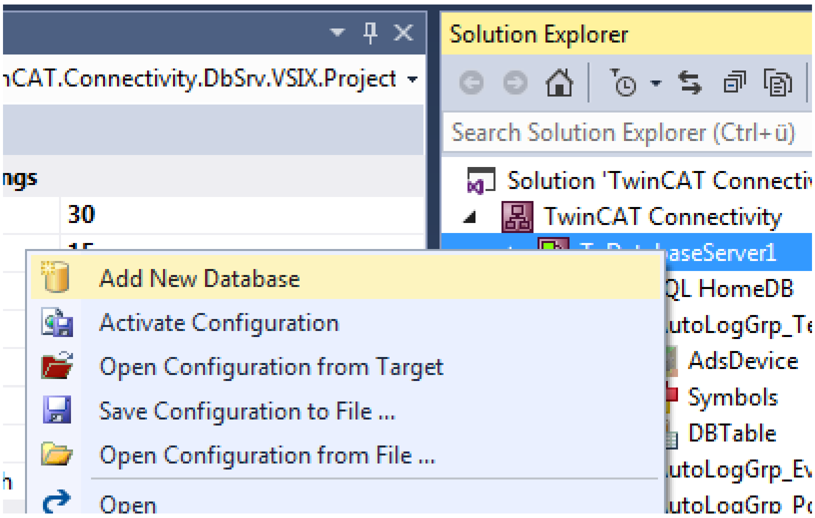
For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

Further configuration settings are available to control the read process from the database. These settings refer to the TwinCAT Database Server on the target system:

MaxStringLength	Maximum string length of the variables in the PLC
MaxByteArrayLength	Maximum byte array length of the variables in the PLC
DBNullAllowed	Indicates whether ZERO values are accepted in the TwinCAT Database Server.
DBConnectionTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection error while attempts are made to establish a connection.
DBCommandTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection fault when a command was sent. If large data quantities are involved, processing of a command may take quite some time, depending on the database and the infrastructure.

Adding a database configuration

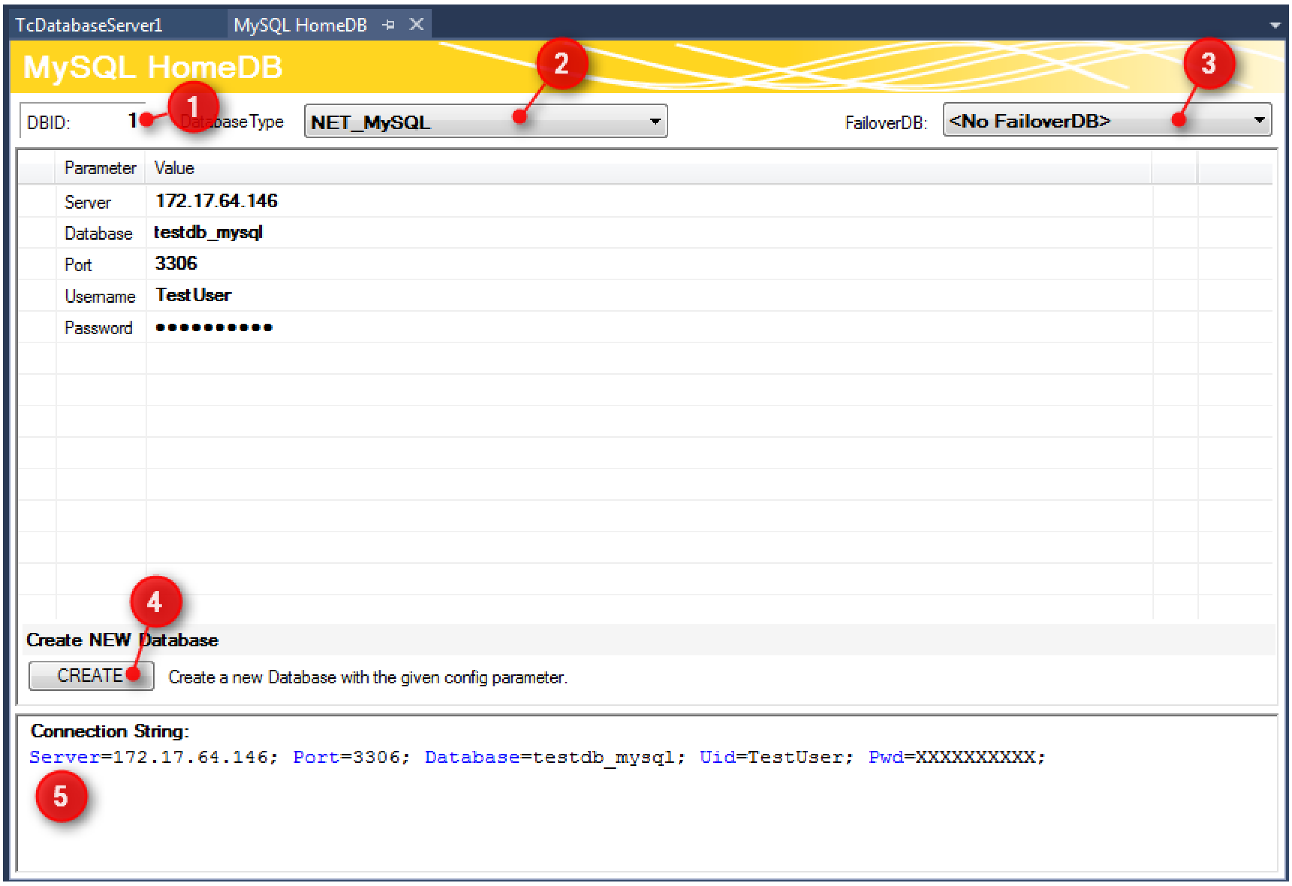
A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.



ıCAT.Connectivity.DbSrv.VSIX.Project
ngs
30
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connecti
TwinCAT Connectivity
T...aseServer1
QL HomeDB
utoLogGrp_Te
AdsDevice
Symbols
DBTable
utoLogGrp_Ev
utoLogGrp_Pc
Add New Database
Activate Configuration
Open Configuration from Target
Save Configuration to File ...
Open Configuration from File ...
Open

A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

Editor for database configurations



MySQL HomeDB

DBID: 1 DatabaseType NET_MySQL FailoverDB: <No FailoverDB>

Parameter	Value
Server	172.17.64.146
Database	testdb_mysql
Port	3306
Username	TestUser
Password	●●●●●●●●●●●

Create NEW Database

CREATE Create a new Database with the given config parameter.

Connection String:

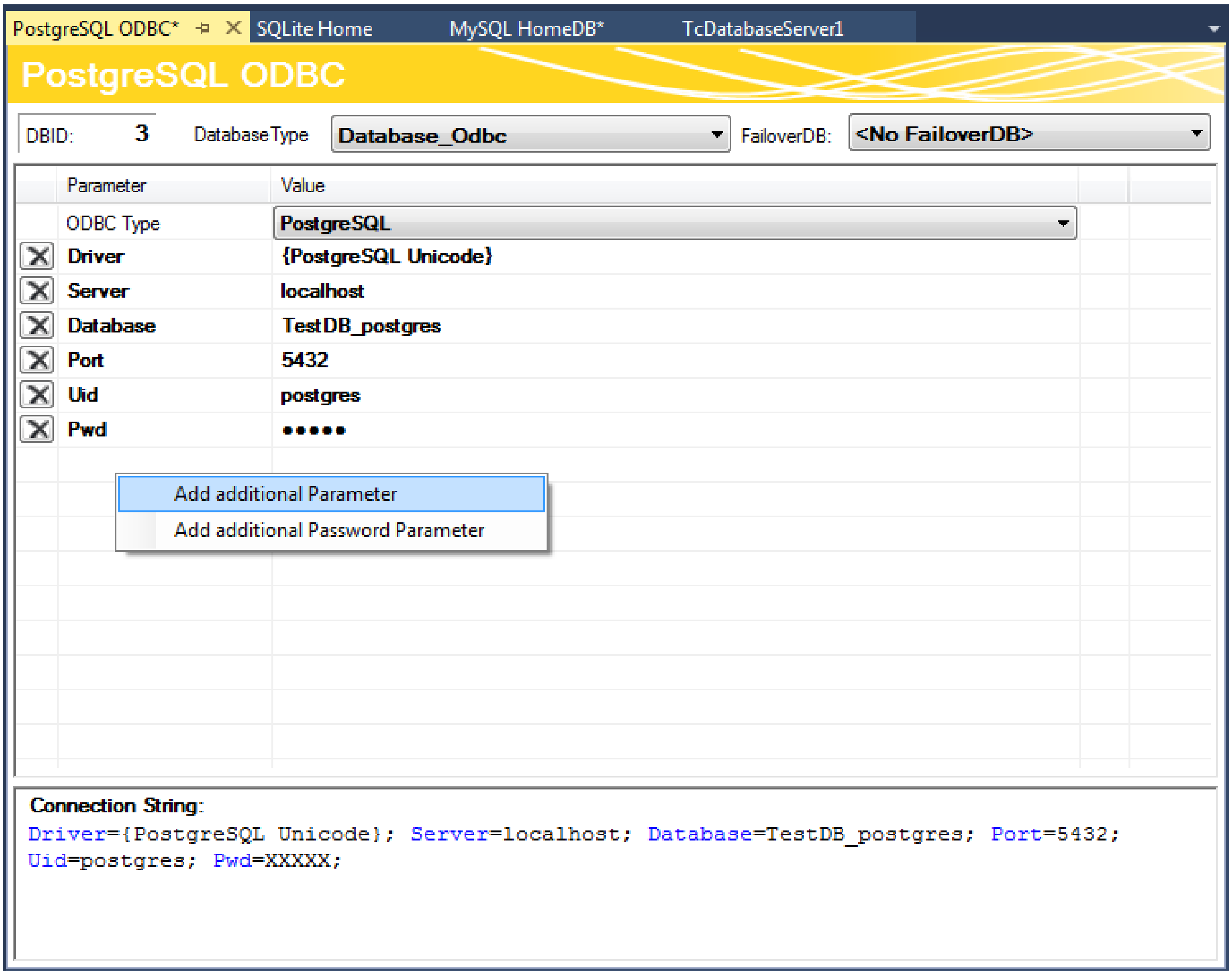
```
Server=172.17.64.146; Port=3306; Database=testdb_mysql; Uid=TestUser; Pwd=XXXXXXXXXX;
```

The database ID, which is required for some function blocks in the PLC, is shown in the upper part of the editor (1). The database types of the target database can be selected from the drop-down menu (2). Another option is the ODBC interface for a database, although this is not yet supported. Note that not all functions of the TwinCAT Database Server can be guaranteed, depending on the database.

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.

For each database [▸ 118] additional adjustable parameters are available. Depending on the database a connection string (5) is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

The **CREATE** (4) button can be used to create a new database. This function is only displayed if the respective database supports it.


PostgreSQL ODBC* SQLite Home MySQL HomeDB* TcDatabaseServer1
PostgreSQL ODBC
Add additional Parameter
Add additional Password Parameter

DBID: 3 DatabaseType Database_Odbc FailoverDB: <No FailoverDB>

Parameter	Value
ODBC Type	PostgreSQL
Driver	{PostgreSQL Unicode}
Server	localhost
Database	TestDB_postgres
Port	5432
Uid	postgres
Pwd	●●●●●

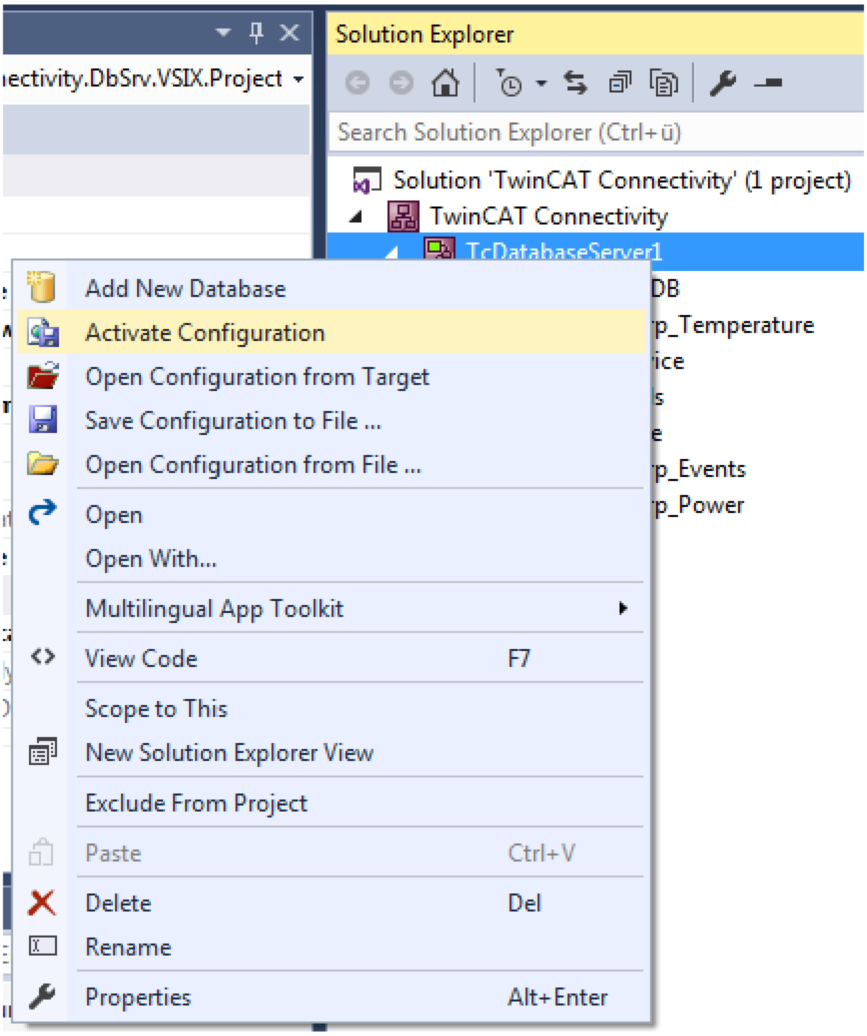
Connection String:

```
Driver={PostgreSQL Unicode}; Server=localhost; Database=TestDB_postgres; Port=5432;
Uid=postgres; Pwd=XXXXX;
```

Unknown databases can be configured via an ODBC interface. In the **ODBC Type** drop-down list select "Unknown Database" and add parameters via the commands in the context menu. They may contain passwords, which are stored in encrypted form. The required connection string can be assembled from these parameters. Note that only limited functions of the TwinCAT Database Server can be used. Only the explicit function blocks of the SQL Expert mode are supported.

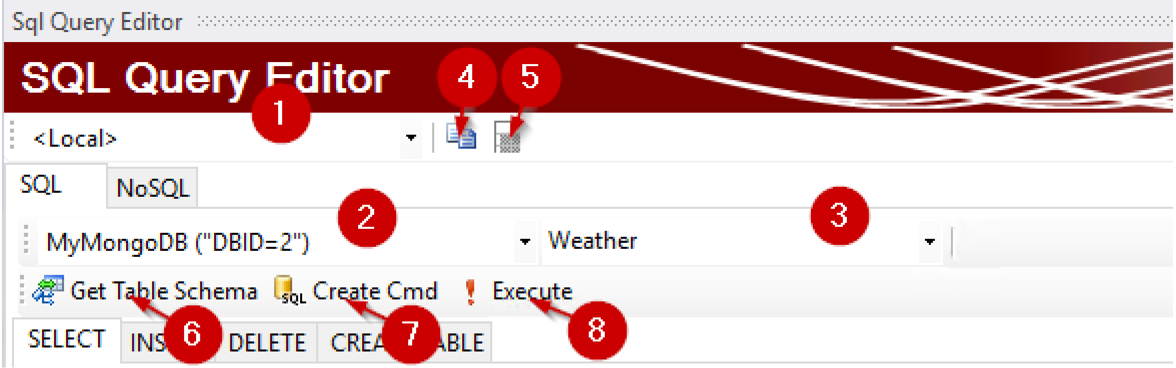
Activating a project

To activate a configured project on the TwinCAT Database Server, use the command **Activate Configuration** in the context menu of the TwinCAT Database Server project.



SQL Query Editor

The SQL Query Editor is a Database Server tool that supports the development of your application. The tool can be used to test connections and SQL commands and to check the compatibility between PLC and databases.



ID	Name	Function
1	Target system	Choose Target System with installed TwinCAT Database Server
2	Database	Selecting the configured database connection
3	Table	Selecting the existing tables in the database
4	Copying for PLC	Copying the SQL command to the PLC string. This can be copied into the PLC source code. Special characters are automatically captured and formatted.
5	Export TC3	Exporting the table schema into a PLC structure. This can be used in the program for SQL commands, for example.
6	Get Table Schema	Reading the table structure
7	Create Cmd	Creating an SQL command, based on the table structure
8	Execute	Executing the SQL command

First select the target system from the routes of your TwinCAT system (1). The TwinCAT Database Server must be installed on the target system. If a NoSQL database is stored in the configuration, an additional NoSQL tab is visible. You will find the documentation in a subitem below.

All configured databases (2) are displayed, once you have activated the database configurations on the target system. You can also select one of the available tables (3) from the database. Based on this table, you can generate SQL commands from the SQL Query Editor and send them to the database. The SQL commands have different syntax, depending on database type.

Three commands are available for generating the individual SQL commands:

- Get Table Schema: Calls up the structure of the selected table.
    - Information such as the column name, PLC data type and size of variables is displayed. The retrieved structure can also be prepared for your PLC application via the commands **Copy for PLC** (4) or **Export TC3** (5).
- Create Cmd: An SQL command is generated in the command text box, depending on the selected tab. The command syntax may differ, depending on the database type. The previously read table schema is used here.
    - The created SQL command can optionally be modified.
- Execute: The SQL command shown in the text box is executed and returns values, if applicable.

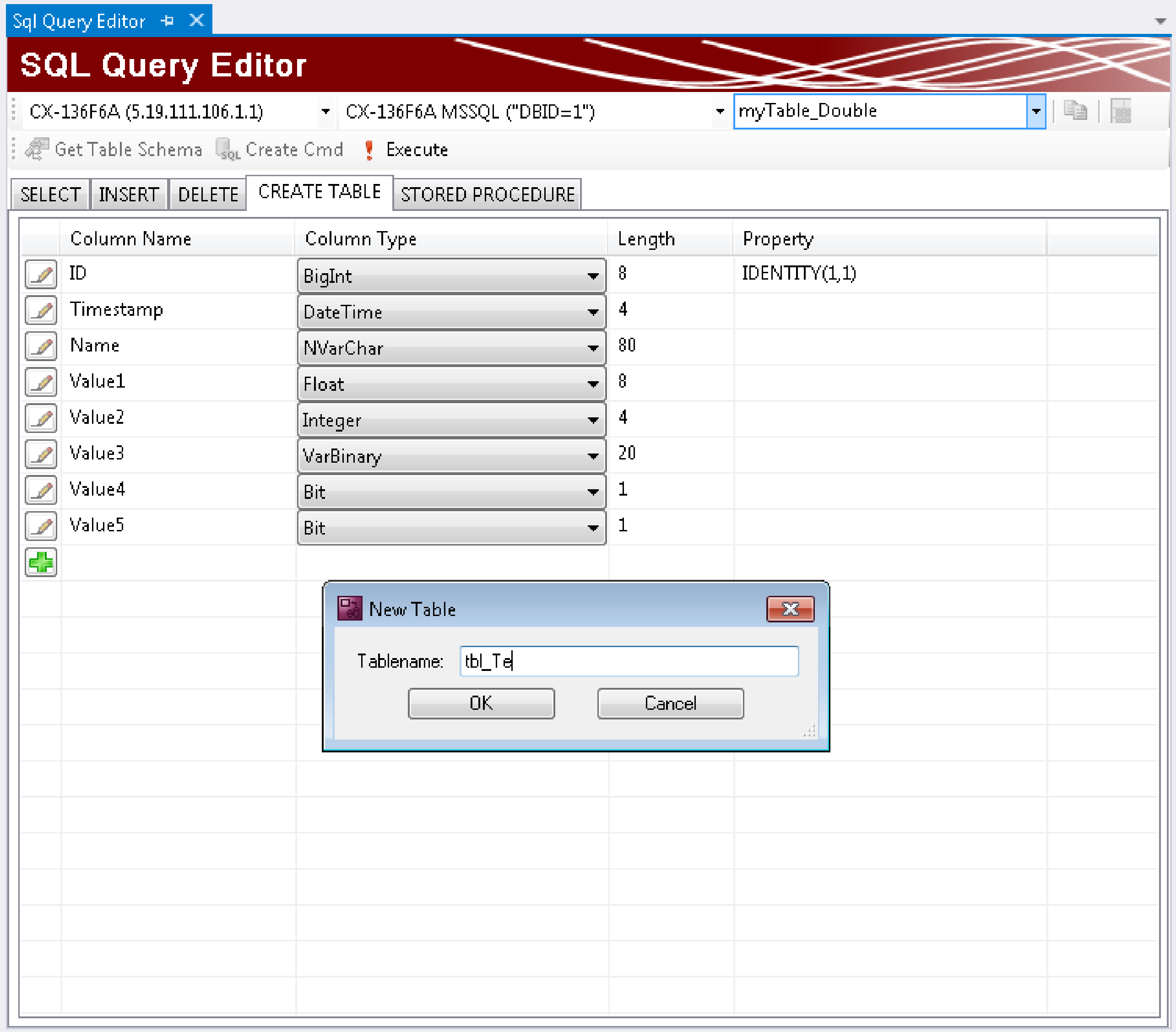
The differences in the individual SQL commands are explained below.

Comment: Since the syntax of SQL commands often collides with the syntax in the ST code of TwinCAT, the SQL Query Editor offers the command "Copy for PLC" (4). The command is used to copy the created and tested SQL commands with the correct formatting for special characters for the ST program code into the cache.

Create Table command

The **CREATE TABLE** tab can be used to create tables within the database. Further columns can be added to the table with (+), as required. Once you have specified the column name and type, you can specify additional properties, in order to generate automatic IDs, for example.

The table name can be determined by executing the command. The table with the configured table structure is created.



CX-136F6A (5.19.111.106.1.1)
CX-136F6A MSSQL ("DBID=1")
myTable_Double

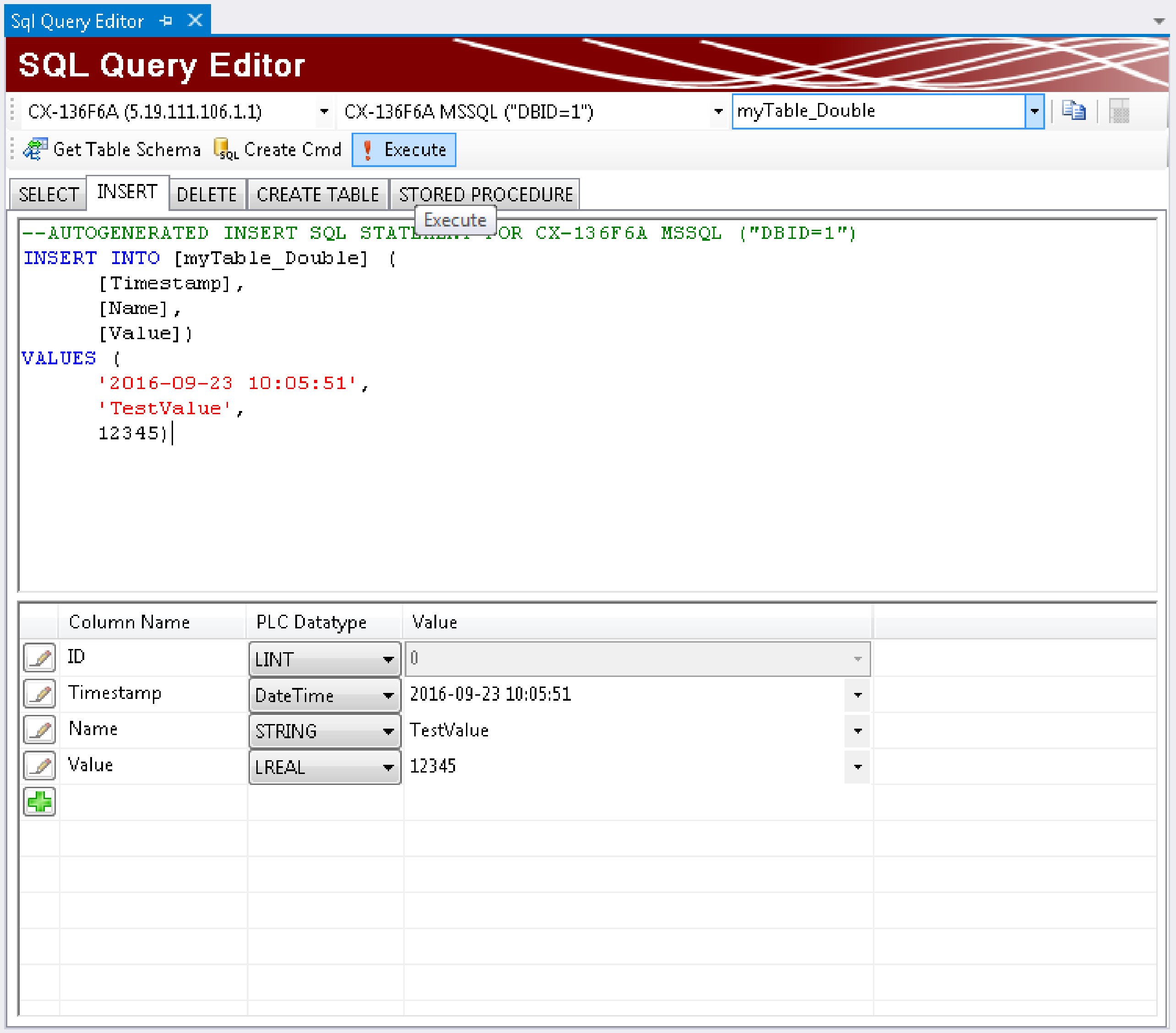
Column Name	Column Type	Length	Property
ID	BigInt	8	IDENTITY(1,1)
Timestamp	DateTime	4	
Name	NVarChar	80	
Value1	Float	8	
Value2	Integer	4	
Value3	VarBinary	20	
Value4	Bit	1	
Value5	Bit	1	

Tablename: tbl_Te

Insert command

The Insert command gives the opportunity to write records into the table. The values under "Value" can be modified once the table structure has been retrieved. If the command is then generated, the values in the Insert command will automatically be in the right format. These values are written into the table when the command is executed.

ℹ This value cannot be customized if automatic ID generation is used.

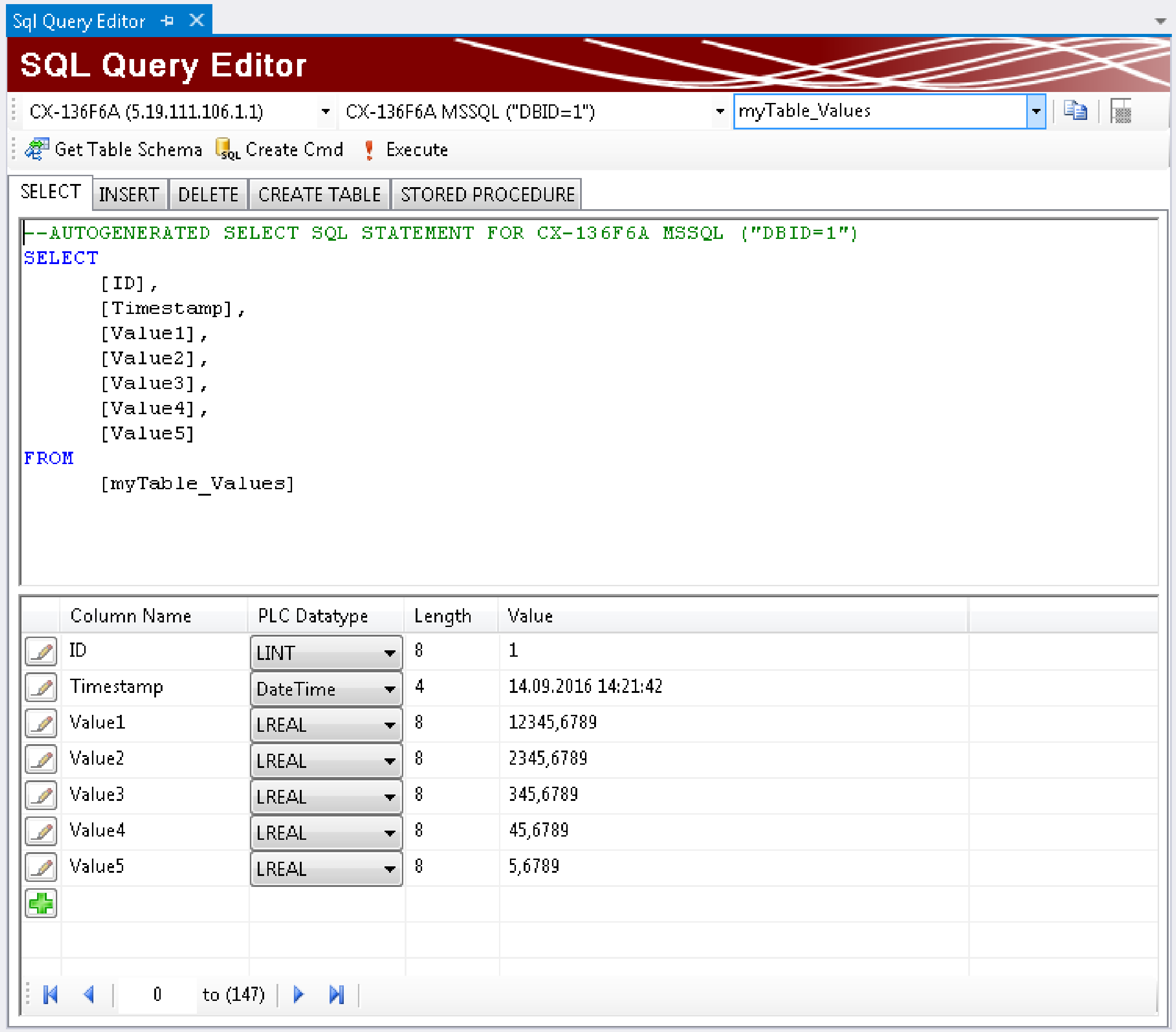


```
--AUTOGENERATED INSERT SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
INSERT INTO [myTable_Double] (
       [Timestamp],
       [Name],
       [Value])
VALUES (
       '2016-09-23 10:05:51',
       'TestValue',
       12345)
```

Column Name	PLC Datatype	Value
ID	LINT	0
Timestamp	DateTime	2016-09-23 10:05:51
Name	STRING	TestValue
Value	LREAL	12345

Select command

Select commands can be created and sent via the **SELECT** tab. Select commands give the opportunity to read records from the databases. After executing the command, values are returned if they exist in the table. They are listed under "Value" in the table structure display. Use the arrows under the display to navigate through the individual records.



CX-136F6A (5.19.111.106.1.1) CX-136F6A MSSQL ("DBID=1") myTable_Values

```
--AUTOGENERATED SELECT SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
SELECT
        [ID],
        [Timestamp],
        [Value1],
        [Value2],
        [Value3],
        [Value4],
        [Value5]
FROM
        [myTable_Values]
```

Column Name	PLC Datatype	Length	Value
ID	LINT	8	1
Timestamp	DateTime	4	14.09.2016 14:21:42
Value1	LREAL	8	12345,6789
Value2	LREAL	8	2345,6789
Value3	LREAL	8	345,6789
Value4	LREAL	8	45,6789
Value5	LREAL	8	5,6789

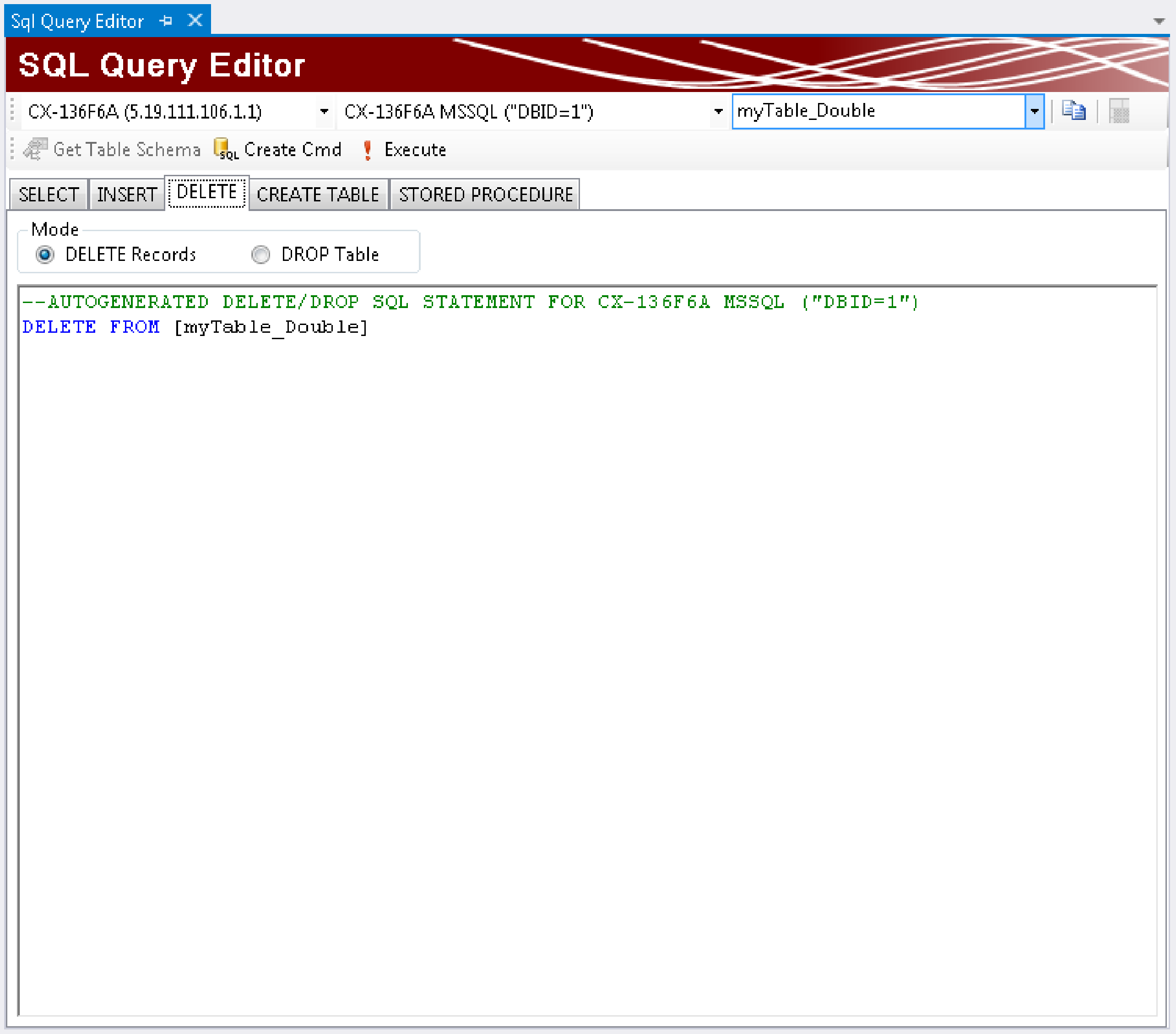
0 to (147)

Delete command

The Delete command has two functions.

1. **DELETE Records**: Deletes the contents of a table.
2. **DROP table:** Deletes the whole table.

This SQL command can also be customized, in order to delete only a particular section of the table, for example.



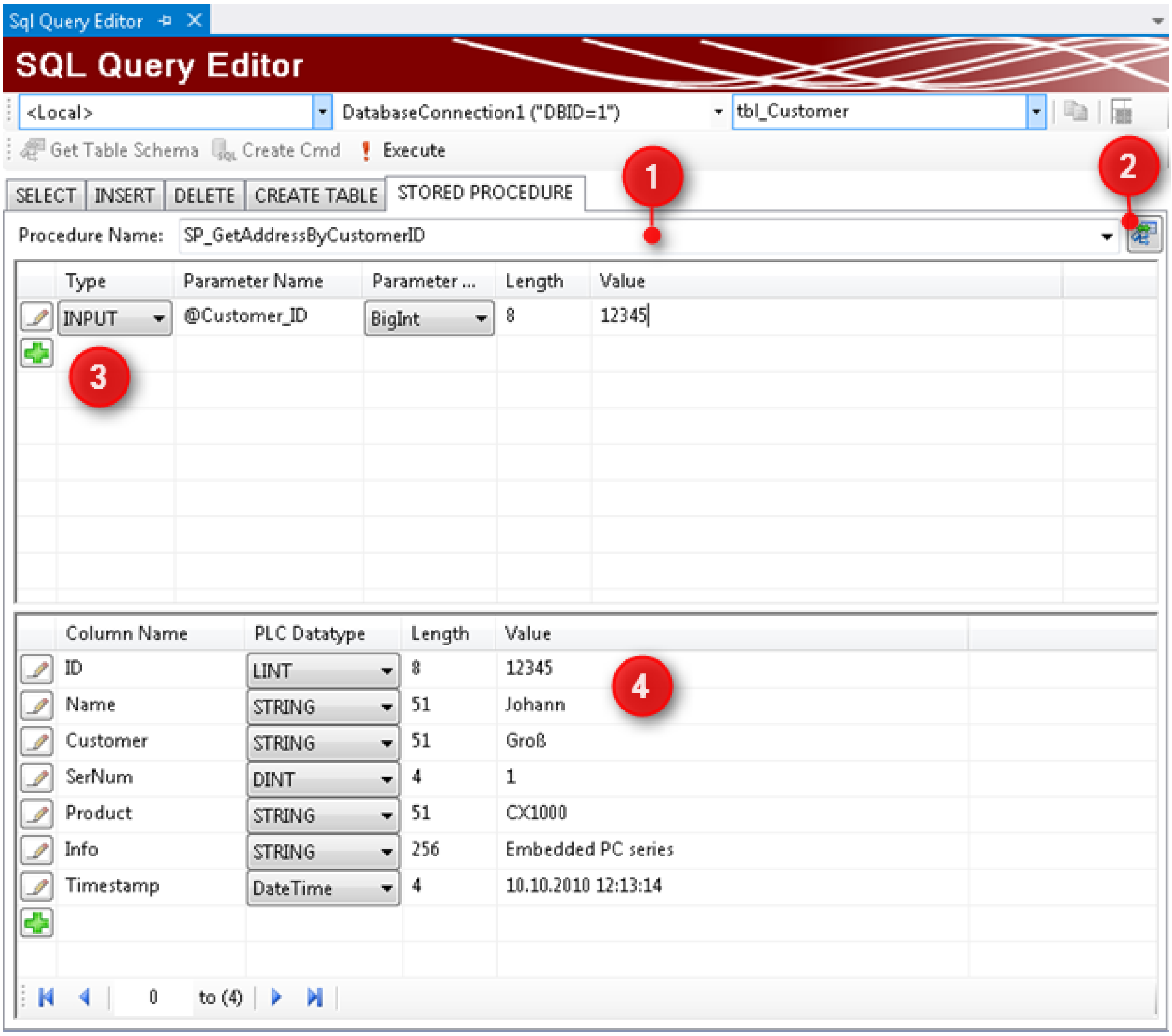
```
--AUTOGENERATED DELETE/DROP SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
DELETE FROM [myTable_Double]
```

Stored Procedures

The TwinCAT Database Server supports "Stored Procedures", which provide numerous databases for processing more complex queries at the database level or to make a simplified interface available.

If Stored Procedures are available in the database and the table, you can list and select them (1). The input and output parameters can be picked up automatically (2) and transferred to the tables in the display (3)(4).

The parameter type, name and data type are displayed there. In addition you can insert values here, in order to execute the Stored Procedures with the input values via "Execute". The result is displayed in the output values (4). If several records are returned, the arrow keys can be used to switch between them. This functionality serves as development aid for the call in the PLC. The results are returned there by calling the corresponding function block [▶ 191].



<Local> DatabaseConnection1 ("DBID=1") tbl_Customer

Procedure Name: SP_GetAddressByCustomerID

Type	Parameter Name	Parameter ...	Length	Value
INPUT	@Customer_ID	BigInt	8	12345

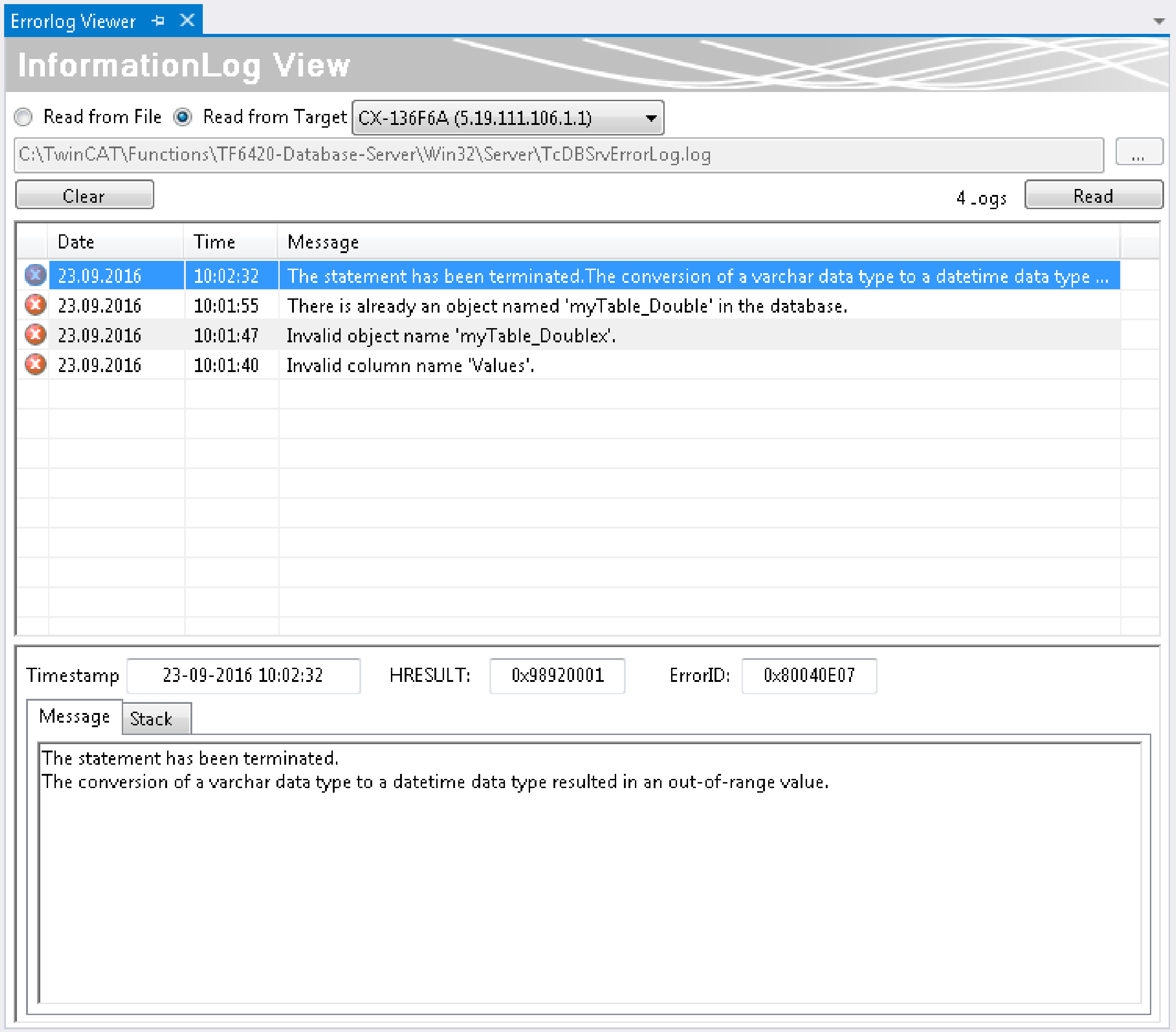
Column Name	PLC Datatype	Length	Value
ID	LINT	8	12345
Name	STRING	51	Johann
Customer	STRING	51	Groß
SerNum	DINT	4	1
Product	STRING	51	CX1000
Info	STRING	256	Embedded PC series
Timestamp	DateTime	4	10.10.2010 12:13:14

InformationLog View for diagnostics

The InformationLog View is available for troubleshooting.

InformationLog View is a tool for reading log files from the TwinCAT Database Server. Recorded information is displayed with a timestamp, IDs and error messages in plain text.

The log files can not only be viewed or emptied via direct file access, but also directly via the target. This is particularly advantageous with distributed Database Servers in a network, for quick and easy access to the log file. For this access a route to the target device must exist.



C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server\TcDBSrvErrorLog.log

Date	Time	Message
23.09.2016	10:02:32	The statement has been terminated.The conversion of a varchar data type to a datetime data type ...
23.09.2016	10:01:55	There is already an object named 'myTable_Double' in the database.
23.09.2016	10:01:47	Invalid object name 'myTable_Doublex'.
23.09.2016	10:01:40	Invalid column name 'Values'.

Timestamp: 23-09-2016 10:02:32 HRESULT: 0x98920001 ErrorID: 0x80040E07

```
The statement has been terminated.
The conversion of a varchar data type to a datetime data type resulted in an out-of-range value.
```

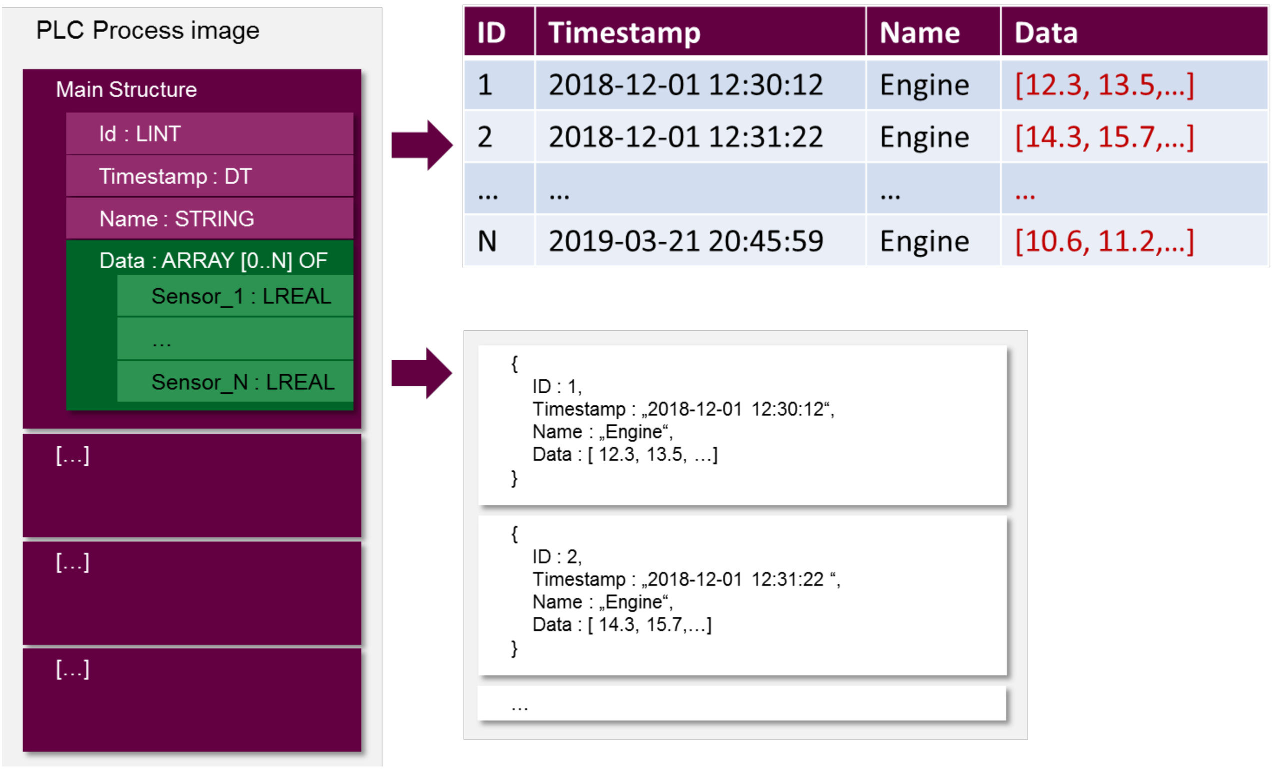
5.1.1.5 NoSql Expert Mode

NoSQL

NoSQL databases (not only Sequel) differ from conventional relational data storage.

Document-based databases:

Records are stored as documents in the database. This offers the advantage of being able to archive data in a more flexible and hierarchical manner.



ID	Timestamp	Name	Data
1	2018-12-01 12:30:12	Engine	[12.3, 13.5,...]
2	2018-12-01 12:31:22	Engine	[14.3, 15.7,...]
...	...	...	...
N	2019-03-21 20:45:59	Engine	[10.6, 11.2,...]

```
{
   ID : 1,
   Timestamp : „2018-12-01  12:30:12“,
   Name : „Engine“,
   Data : [ 12.3,  13.5,  …]
}
```

```
{
   ID : 2,
   Timestamp : „2018-12-01  12:31:22 “,
   Name : „Engine“,
   Data : [ 14.3,  15.7,…]
}
```

```
…
```

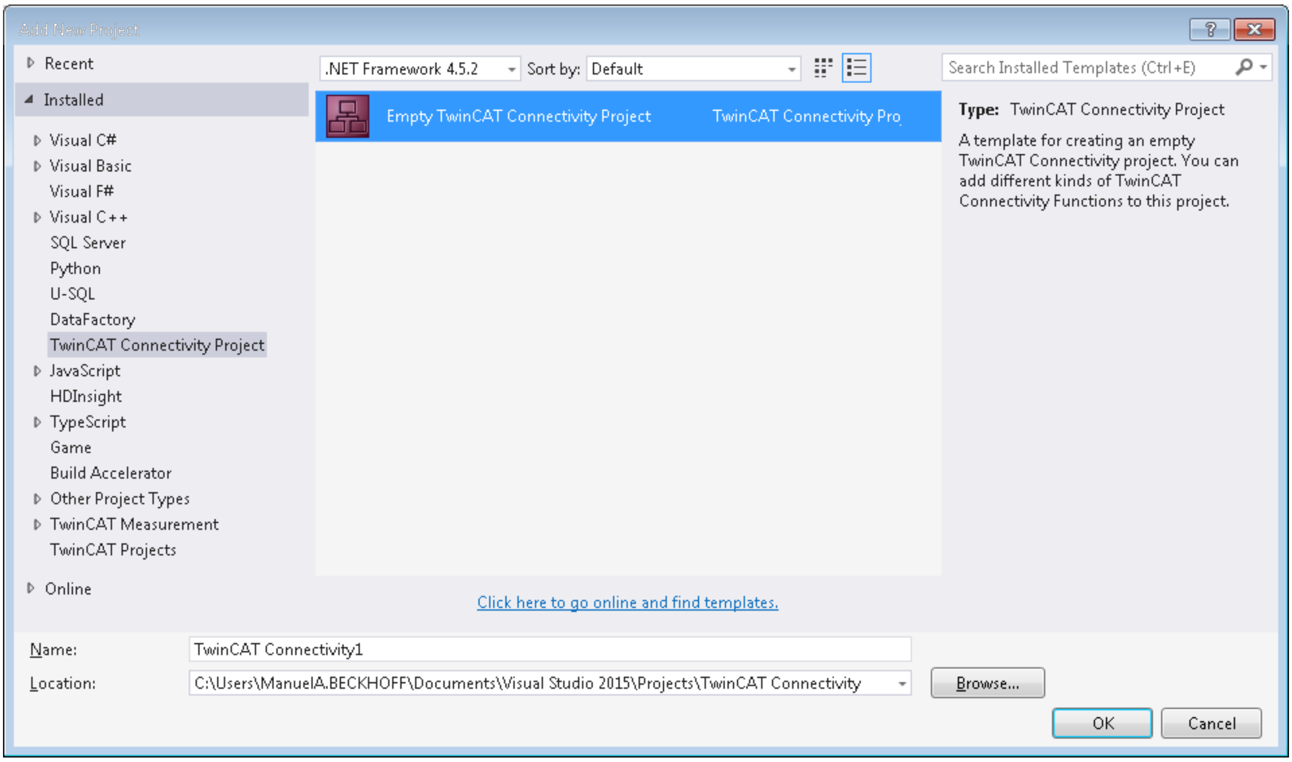
If a record consists of more than one flat structure of basic data types, it can no longer be mapped directly via a relational database. NoSQL databases offer this flexibility. Changes in the program code and the corresponding structures can also be easily adopted without having to create a new table.

Document-based databases usually save the data as JSON-formatted records. The records can all be different.

Build Project

The TwinCAT Connectivity extension for Visual Studio provides a new project template. When a new project is created, the **TwinCAT Connectivity Project** category appears as an option.

To create a new TwinCAT Connectivity project, select **Empty TwinCAT Connectivity Project**, specify the project name and the storage location and click **OK** to add it to the solution. In this way, TwinCAT Connectivity projects or TwinCAT Database Server projects can conveniently be created in parallel with TwinCAT or other Visual Studio projects.



.NET Framework 4.5.2

Sort by: Default

Type: TwinCAT Connectivity Project

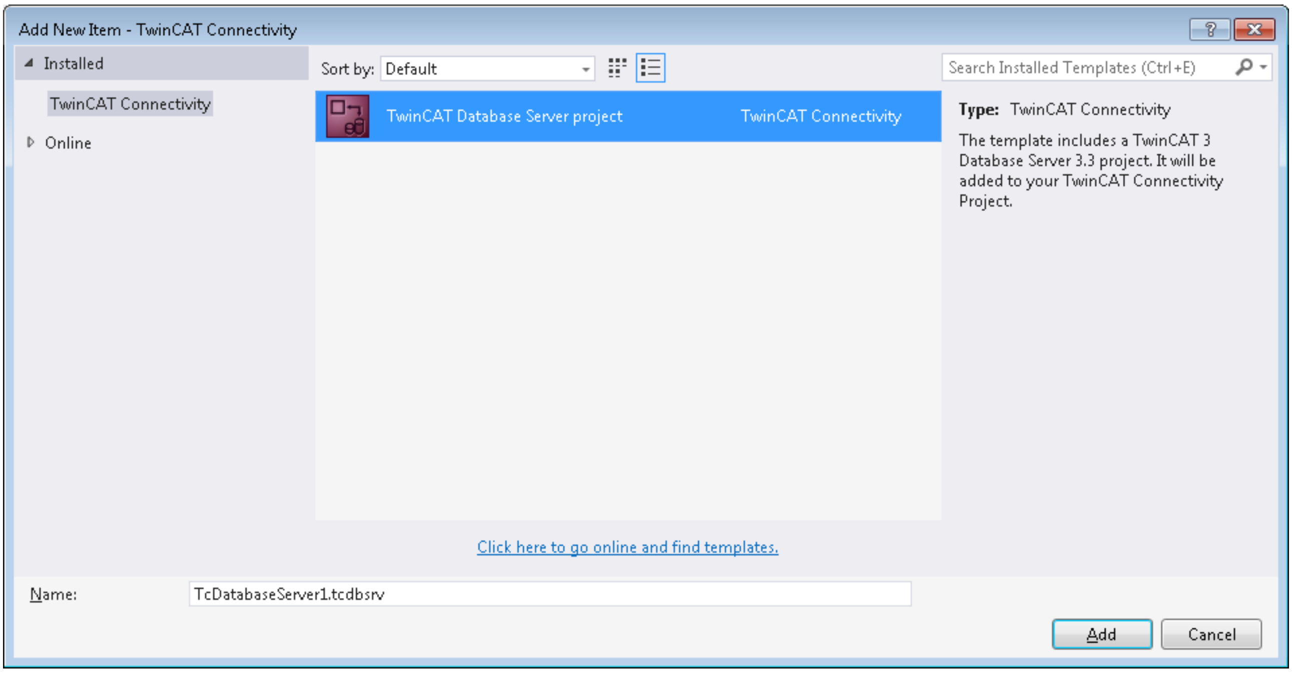
A template for creating an empty TwinCAT Connectivity project. You can add different kinds of TwinCAT Connectivity Functions to this project.

Name: TwinCAT Connectivity1

Location: C:\Users\ManuelA.BECKHOFF\Documents\Visual Studio 2015\Projects\TwinCAT Connectivity

A new project node appears in the solution. Below the Connectivity project node you can add subprojects for the supported connectivity functions.

Use **Add** to add a new TwinCAT Database Server project to the TwinCAT Connectivity project. The TwinCAT Database Server project can be found in the list of existing Item Templates.


Add New Item - TwinCAT Connectivity
Installed
TwinCAT Connectivity
Online
Search Installed Templates (Ctrl+E)
TwinCAT Database Server project
TwinCAT Connectivity
Click here to go online and find templates.
Add
Cancel

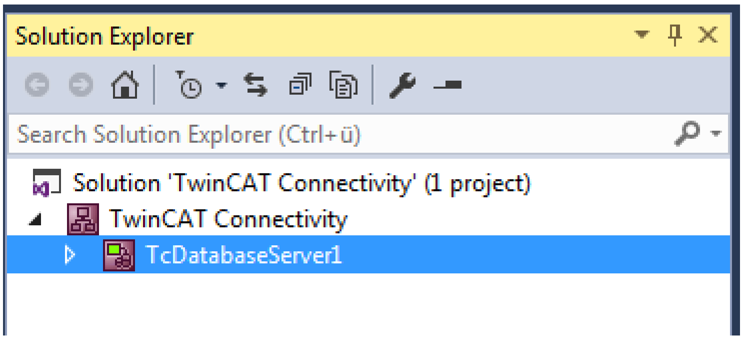
Sort by: Default

Type: TwinCAT Connectivity

The template includes a TwinCAT 3 Database Server 3.3 project. It will be added to your TwinCAT Connectivity Project.

Name: TcDatabaseServer1.tcdbsrv

A new TwinCAT Database Server project is created under the TwinCAT Connectivity node.



Solution Explorer

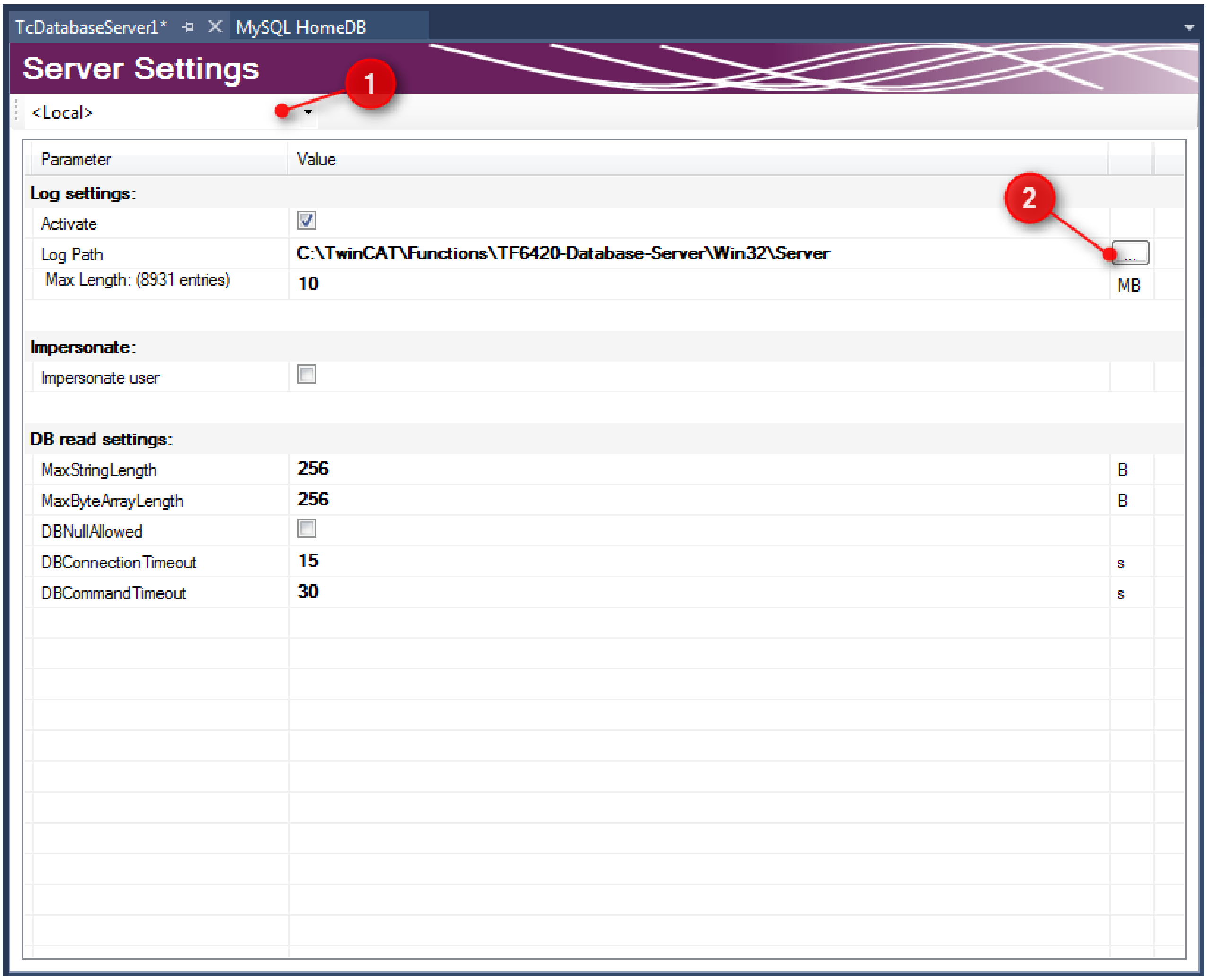
Search Solution Explorer (Ctrl+ü)

Solution 'TwinCAT Connectivity' (1 project)
- TwinCAT Connectivity
  - TcDatabaseServer1

This is now used as the basis for the pending configuration of a TwinCAT Database Server. The document can be edited either via the Properties window or via an editor.

A Connectivity project can be associated with any number of TwinCAT Database Server projects or other projects, and it may therefore contain several configurations.

Editor for server settings



Parameter	Value	
Log settings:		
Activate	☑	
Log Path	C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server	...
Max Length: (8931 entries)	10	MB
Impersonate:		
Impersonate user	☐	
DB read settings:		
MaxStringLength	256	B
MaxByteArrayLength	256	B
DBNullAllowed	☐	
DBConnectionTimeout	15	s
DBCommandTimeout	30	s

The **Server Settings** editor can be used to edit the settings for the TwinCAT Database Server. These are general settings relating to the corresponding server. In the drop-down menu (1) you can select the target system via the Ams NetID. To this end you have to create a route to the target system via TwinCAT. When a finished configuration is transferred, the settings are stored in the TwinCAT Database Server for this target system.

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the Information Log Viewer [▶ 53]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.

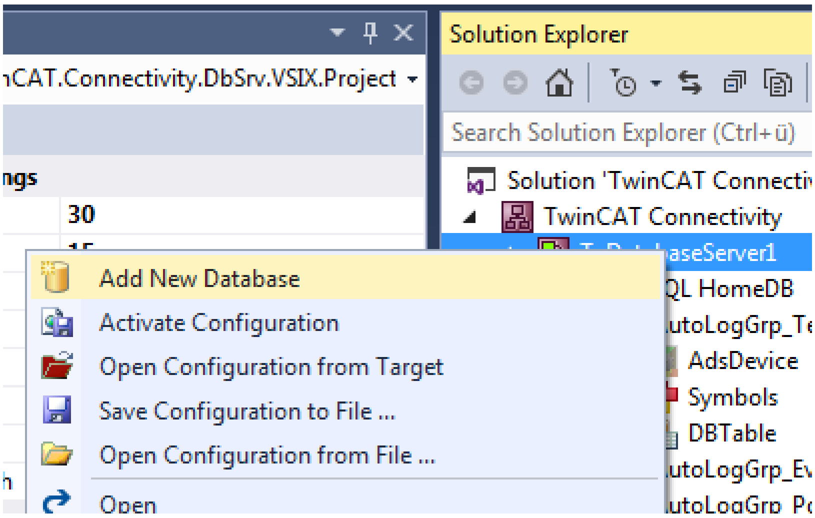
For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

Further configuration settings are available to control the read process from the database. These settings refer to the TwinCAT Database Server on the target system:

MaxStringLength	Maximum string length of the variables in the PLC
MaxByteArrayLength	Maximum byte array length of the variables in the PLC
DBNullAllowed	Indicates whether ZERO values are accepted in the TwinCAT Database Server.
DBConnectionTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection error while attempts are made to establish a connection.
DBCommandTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection fault when a command was sent. If large data quantities are involved, processing of a command may take quite some time, depending on the database and the infrastructure.

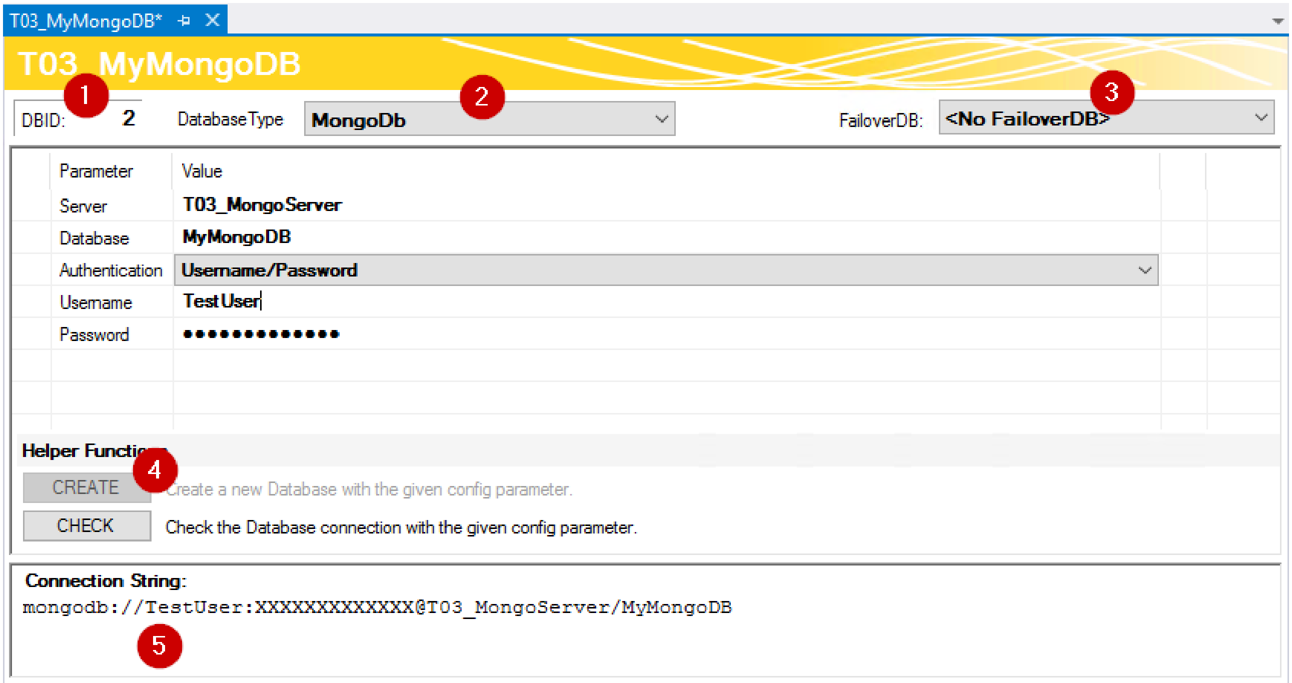
Adding a database configuration

A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.


ıCAT.Connectivity.DbSrv.VSIX.Project
ngs
30
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TwinCAT Connecti
TwinCAT Connectivity
aseServer1
QL HomeDB
utoLogGrp_Te
AdsDevice
Symbols
DBTable
utoLogGrp_Ev
utoLogGrp_Pc
Add New Database
Activate Configuration
Open Configuration from Target
Save Configuration to File ...
Open Configuration from File ...
Open

A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

Editor for database configurations



DBID: 2 DatabaseType MongoDb FailoverDB: <No FailoverDB>

Parameter	Value
Server	T03_MongoServer
Database	MyMongoDB
Authentication	Username/Password
Username	TestUser
Password	●●●●●●●●●●●●●

Create a new Database with the given config parameter.

Check the Database connection with the given config parameter.

Connection String:

```
mongodb://TestUser:XXXXXXXXXXXXX@T03_MongoServer/MyMongoDB
```

The database ID, which is required for some function blocks in the PLC, is shown in the upper part of the editor (1). The database types of the target database can be selected from the drop-down menu (2). Another option is the ODBC interface for a database, although this is not yet supported. Note that not all functions of the TwinCAT Database Server can be guaranteed, depending on the database.

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.

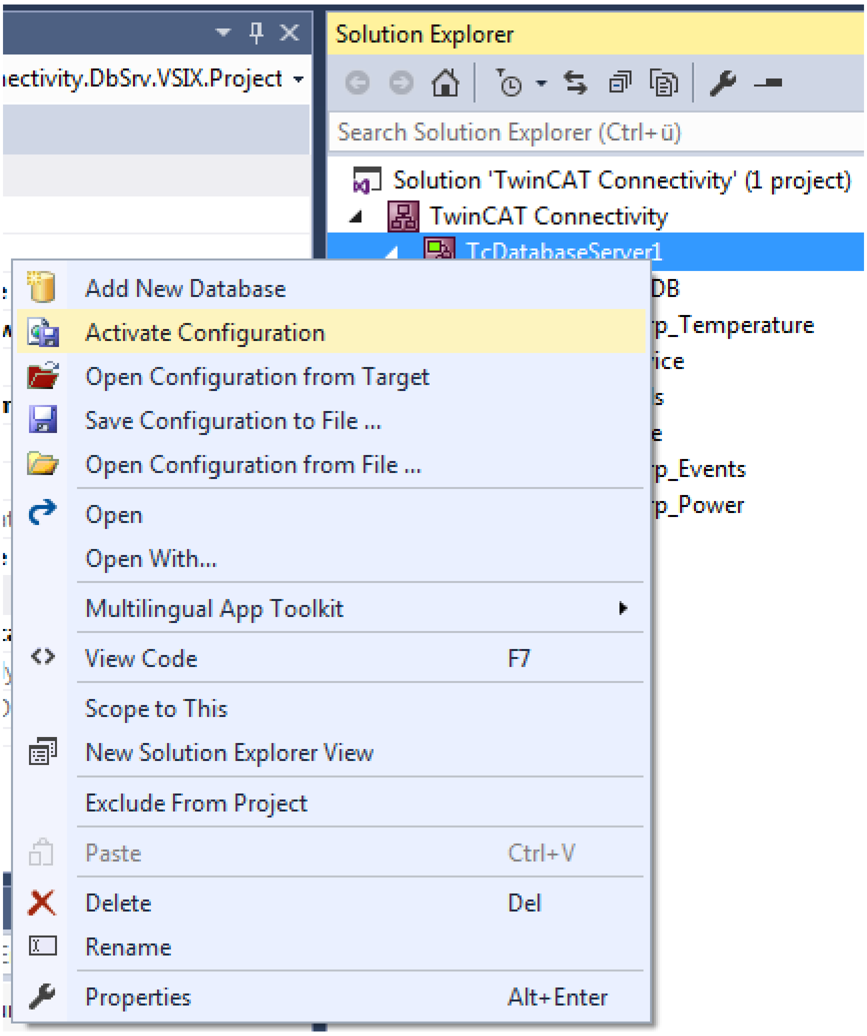
For each database [▶ 118] additional adjustable parameters are available. Depending on the database a connection string (5) is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

The **CREATE** (4) button can be used to create a new database. This function is only displayed if the respective database supports it. **CHECK** can be used to check the connection to the database.

NoSQL databases can also be selected from the target databases. If you activate the project with a NoSQL database, a NoSQL tab is enabled in the SQL query editor to facilitate the use of NoSQL-specific functions.

Activating a project

To activate a configured project on the TwinCAT Database Server, use the command **Activate Configuration** in the context menu of the TwinCAT Database Server project.



MongoDB in PLC Expert mode

PLC Expert mode uses the predefined schema of a database in its function blocks. Normally, the schema of the structures used will not change during operation. In order to nevertheless be able to use the function blocks, the TwinCAT 3 Database Server requires a description of the table schema. A table is simulated. To do this, use the SQL Query Editor to create a table or, in this case, a collection. In addition, unlike for relational databases, an entry is created in a metadata collection. Information on the table schema for the TwinCAT 3 Database Server is stored here.

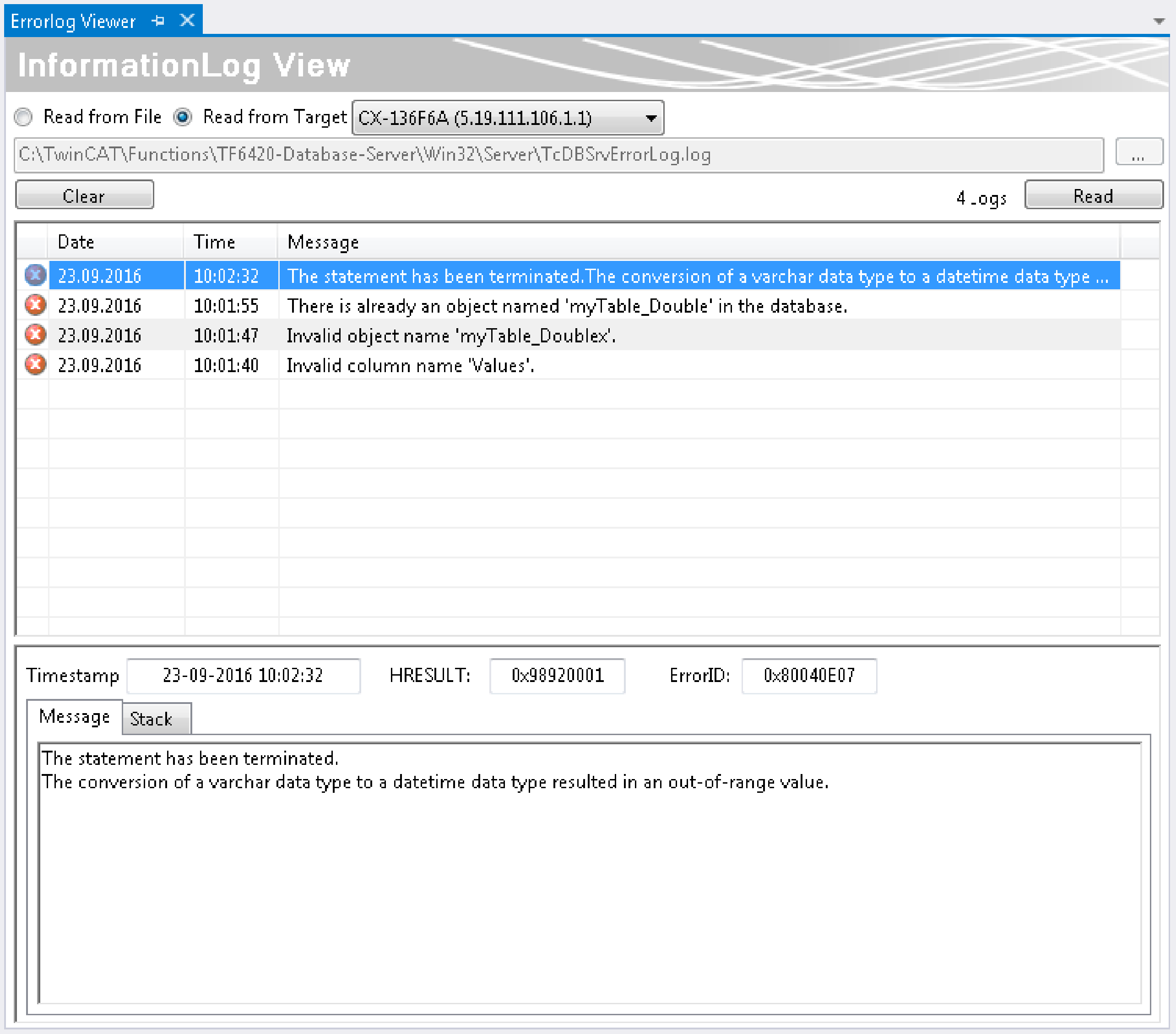
In order to use advanced functionality, e.g. structures of any hierarchy or flexible records, we recommend using the NoSQL function blocks.

InformationLog View for diagnostics

The InformationLog View is available for troubleshooting.

InformationLog View is a tool for reading log files from the TwinCAT Database Server. Recorded information is displayed with a timestamp, IDs and error messages in plain text.

The log files can not only be viewed or emptied via direct file access, but also directly via the target. This is particularly advantageous with distributed Database Servers in a network, for quick and easy access to the log file. For this access a route to the target device must exist.



○ Read from File ● Read from Target CX-136F6A (5.19.111.106.1.1)

C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server\TcDBSrvErrorLog.log

4 Logs

Date	Time	Message
23.09.2016	10:02:32	The statement has been terminated.The conversion of a varchar data type to a datetime data type ...
23.09.2016	10:01:55	There is already an object named 'myTable_Double' in the database.
23.09.2016	10:01:47	Invalid object name 'myTable_Doublex'.
23.09.2016	10:01:40	Invalid column name 'Values'.

Timestamp 23-09-2016 10:02:32 HRESULT: 0x98920001 ErrorID: 0x80040E07

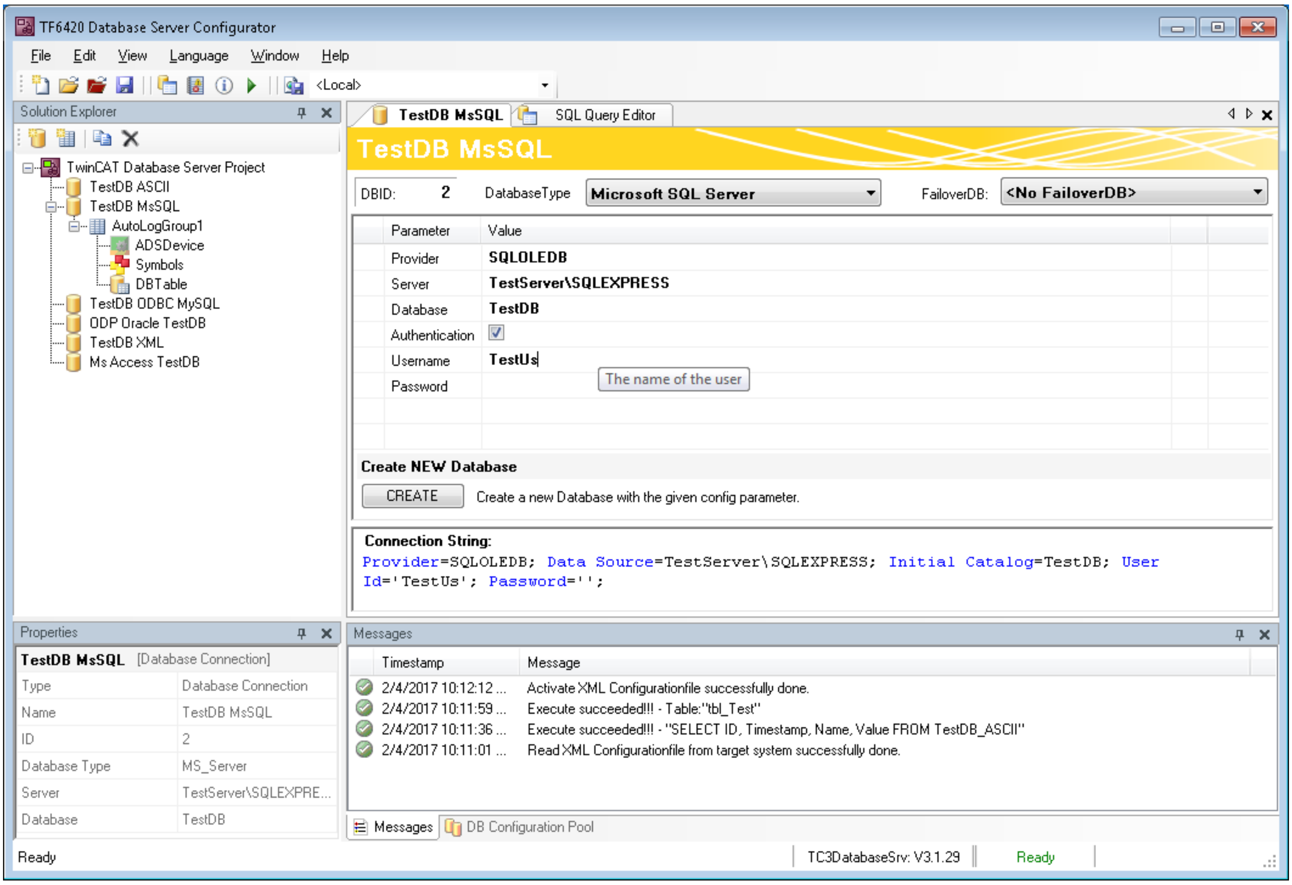
The statement has been terminated.
The conversion of a varchar data type to a datetime data type resulted in an out-of-range value.

5.1.2 Standalone Configurator

5.1.2.1 General

5.1.2.1.1 Interface and basic functions

The TwinCAT 3 Database Server is configured via an XML configuration file.
The settings in the configuration file can easily be created and modified with the help of the XML configuration file editor. New configuration files can be created, and existing configuration files can be read and revised.



TestDB MsSQL

DBID: 2 DatabaseType: Microsoft SQL Server FailoverDB: <No FailoverDB>

Parameter	Value
Provider	SQLOLEDB
Server	TestServer\SQLEXPRESS
Database	TestDB
Authentication	☑
Username	TestUs
Password	

The name of the user

Create NEW Database

CREATE Create a new Database with the given config parameter.

Connection String:

```
Provider=SQLOLEDB; Data Source=TestServer\SQLEXPRESS; Initial Catalog=TestDB; User
Id='TestUs'; Password='';
```

Properties

TestDB MsSQL [Database Connection]

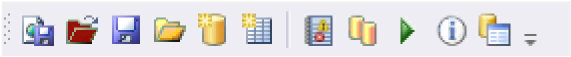
Type	Database Connection
Name	TestDB MsSQL
ID	2
Database Type	MS_Server
Server	TestServer\SQLEXPRE...
Database	TestDB

Messages

Timestamp	Message
2/4/2017 10:12:12 ...	Activate XML Configurationfile successfully done.
2/4/2017 10:11:59 ...	Execute succeeded!!! - Table:"tbl_Test"
2/4/2017 10:11:36 ...	Execute succeeded!!! - "SELECT ID, Timestamp, Name, Value FROM TestDB_ASCII"
2/4/2017 10:11:01 ...	Read XML Configurationfile from target system successfully done.

Toolbar and commands

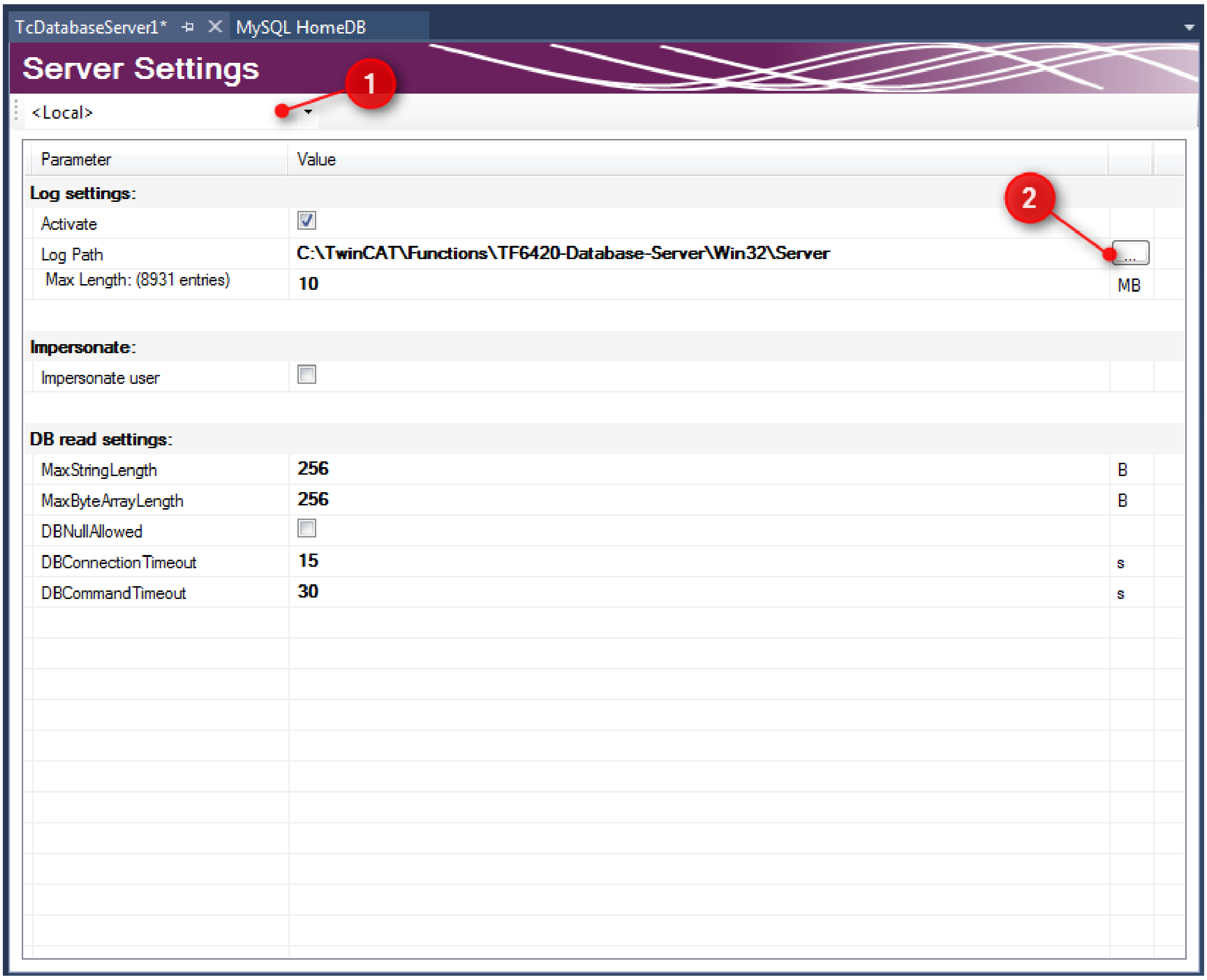
The toolbar has the following elements:



Toolstrip button	Description
	Activation of the configuration
📂	Read configuration of the target device
💾	Save configuration in an XML file
📂	Read configuration from an XML file
	Add new database configuration
	Add new AutoLog group
	Event display
	Database pool
▶	AutoLog Viewer
ⓘ	InformationLog View
	SQL Query Editor

5.1.2.1.2 Project properties

Editor for server settings



Server Settings

<Local>

Parameter	Value	
Log settings:		
Activate	☑	
Log Path	C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server	...
Max Length: (8931 entries)	10	MB
Impersonate:		
Impersonate user	□	
DB read settings:		
MaxStringLength	256	B
MaxByteArrayLength	256	B
DBNullAllowed	□	
DBConnectionTimeout	15	s
DBCommandTimeout	30	s

The **Server Settings** editor can be used to edit the settings for the TwinCAT Database Server. These are general settings relating to the corresponding server. In the drop-down menu (1) you can select the target system via the Ams NetID. To this end you have to create a route to the target system via TwinCAT. When a finished configuration is transferred, the settings are stored in the TwinCAT Database Server for this target system.

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the Information Log Viewer [▶ 53]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.

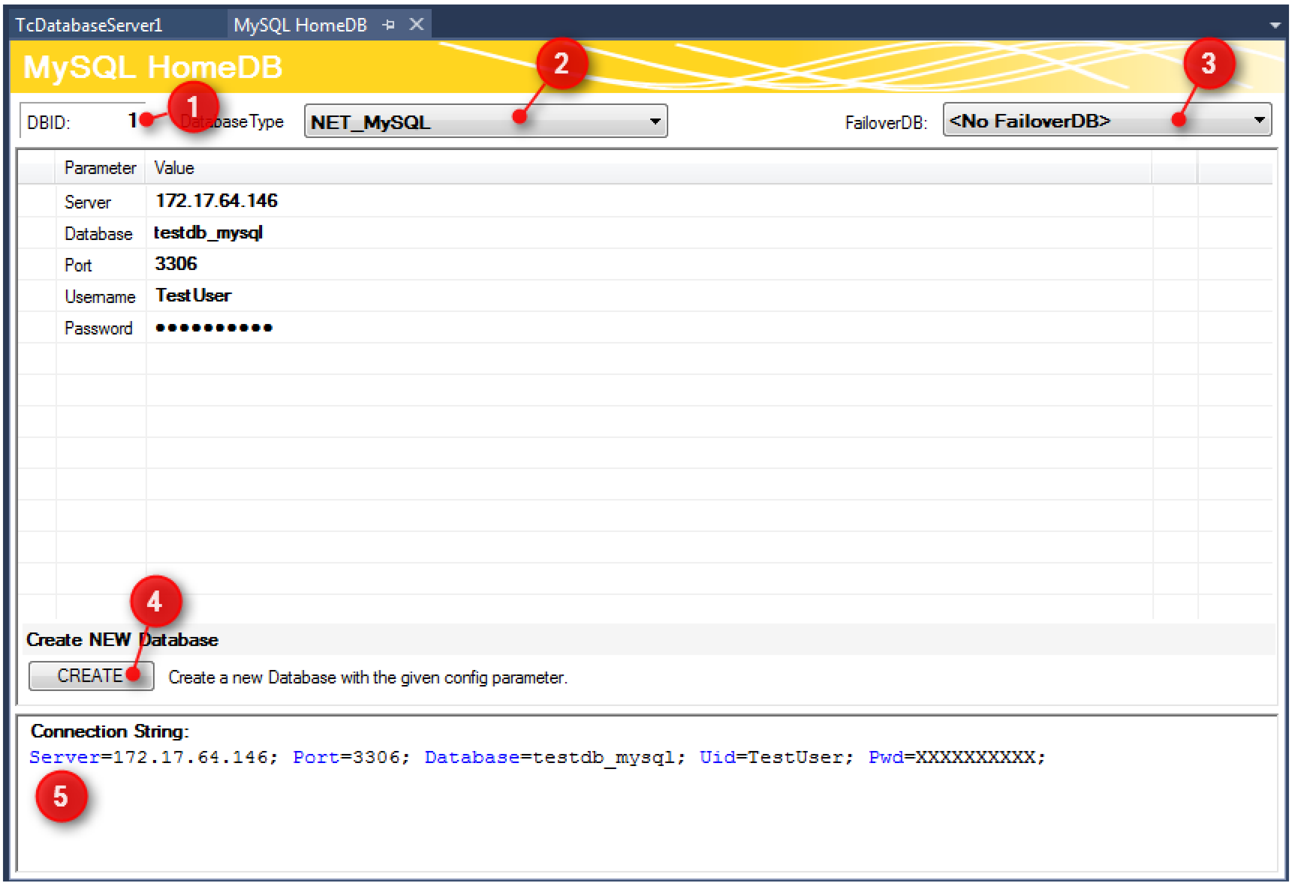
For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

Further configuration settings are available to control the read process from the database. These settings refer to the TwinCAT Database Server on the target system:

MaxStringLength	Maximum string length of the variables in the PLC
MaxByteArrayLength	Maximum byte array length of the variables in the PLC
DBNullAllowed	Indicates whether ZERO values are accepted in the TwinCAT Database Server.
DBConnectionTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection error while attempts are made to establish a connection.
DBCommandTimeout	Indicates the time after which the TwinCAT Database Server assumes a connection fault when a command was sent. If large data quantities are involved, processing of a command may take quite some time, depending on the database and the infrastructure.

5.1.2.1.3 Configuring databases

Editor for database configurations



DBID: 1 DatabaseType NET_MySQL FailoverDB: <No FailoverDB>

Parameter	Value
Server	172.17.64.146
Database	testdb_mysql
Port	3306
Username	TestUser
Password	●●●●●●●●●●

Create NEW Database

CREATE Create a new Database with the given config parameter.

Connection String:

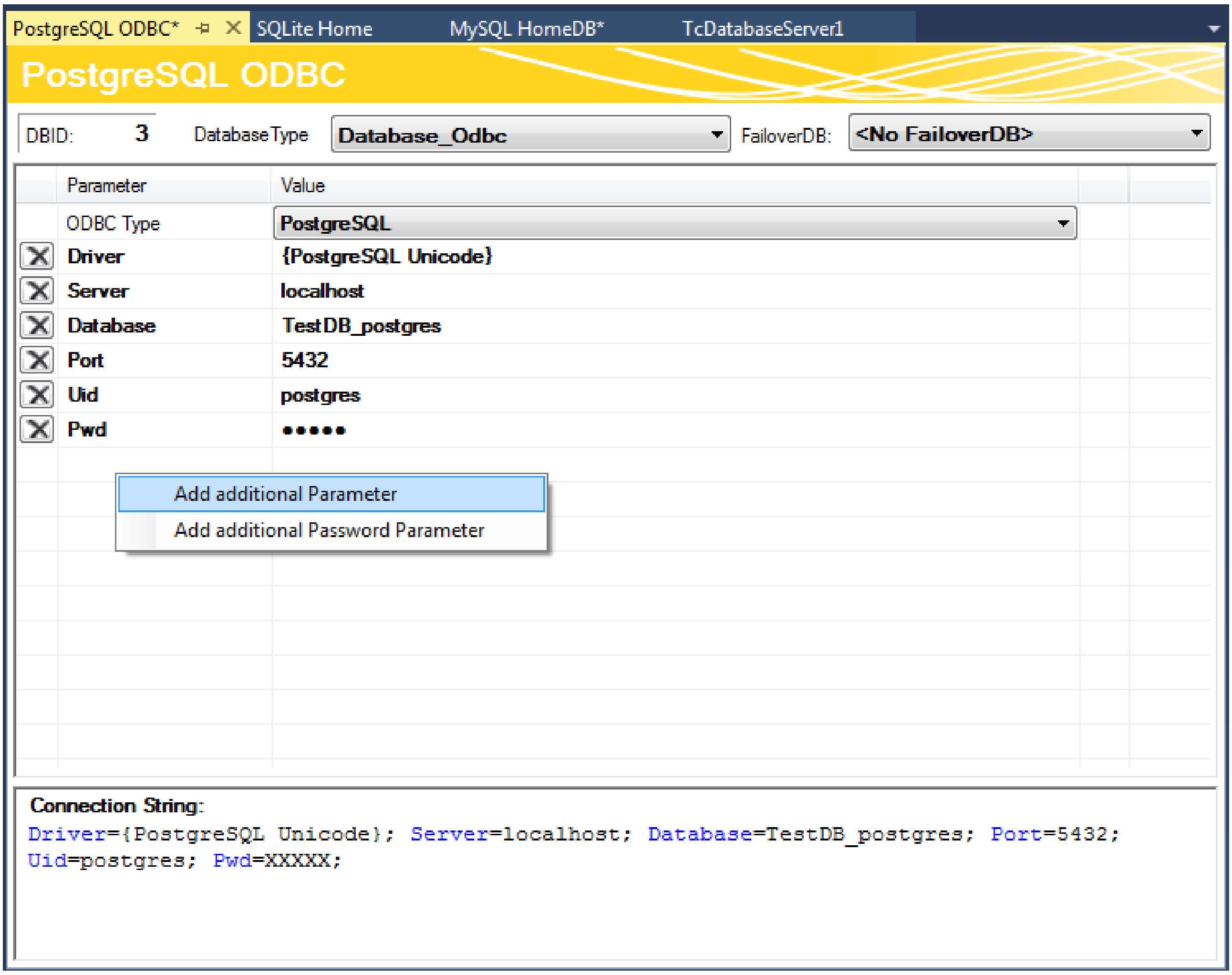
```
Server=172.17.64.146; Port=3306; Database=testdb_mysql; Uid=TestUser; Pwd=XXXXXXXXXX;
```

The database ID, which is required for some function blocks in the PLC, is shown in the upper part of the editor (1). The database types of the target database can be selected from the drop-down menu (2). Another option is the ODBC interface for a database, although this is not yet supported. Note that not all functions of the TwinCAT Database Server can be guaranteed, depending on the database.

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.

For each database [▶ 118] additional adjustable parameters are available. Depending on the database a connection string (5) is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

The **CREATE** (4) button can be used to create a new database. This function is only displayed if the respective database supports it.



DBID: 3 DatabaseType Database_Odbc FailoverDB: <No FailoverDB>

Parameter	Value
ODBC Type	PostgreSQL
Driver	{PostgreSQL Unicode}
Server	localhost
Database	TestDB_postgres
Port	5432
Uid	postgres
Pwd	●●●●●

Connection String:

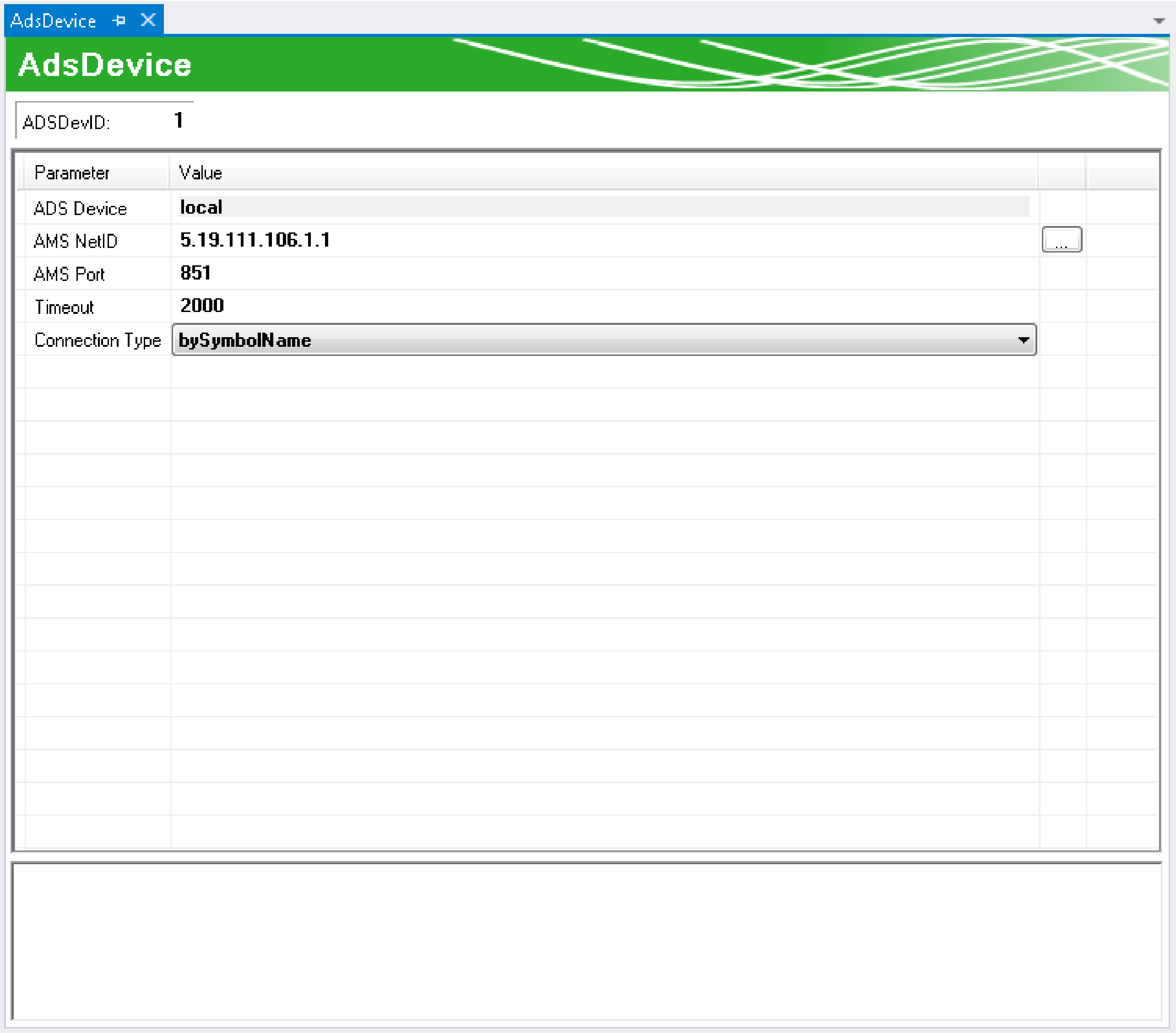
```
Driver={PostgreSQL Unicode}; Server=localhost; Database=TestDB_postgres; Port=5432;
Uid=postgres; Pwd=XXXXX;
```

Unknown databases can be configured via an ODBC interface. In the **ODBC Type** drop-down list select "Unknown Database" and add parameters via the commands in the context menu. They may contain passwords, which are stored in encrypted form. The required connection string can be assembled from these parameters. Note that only limited functions of the TwinCAT Database Server can be used. Only the explicit function blocks of the SQL Expert mode are supported.

5.1.2.1.4 Configuring AutoLog groups

Configuring the ADS device

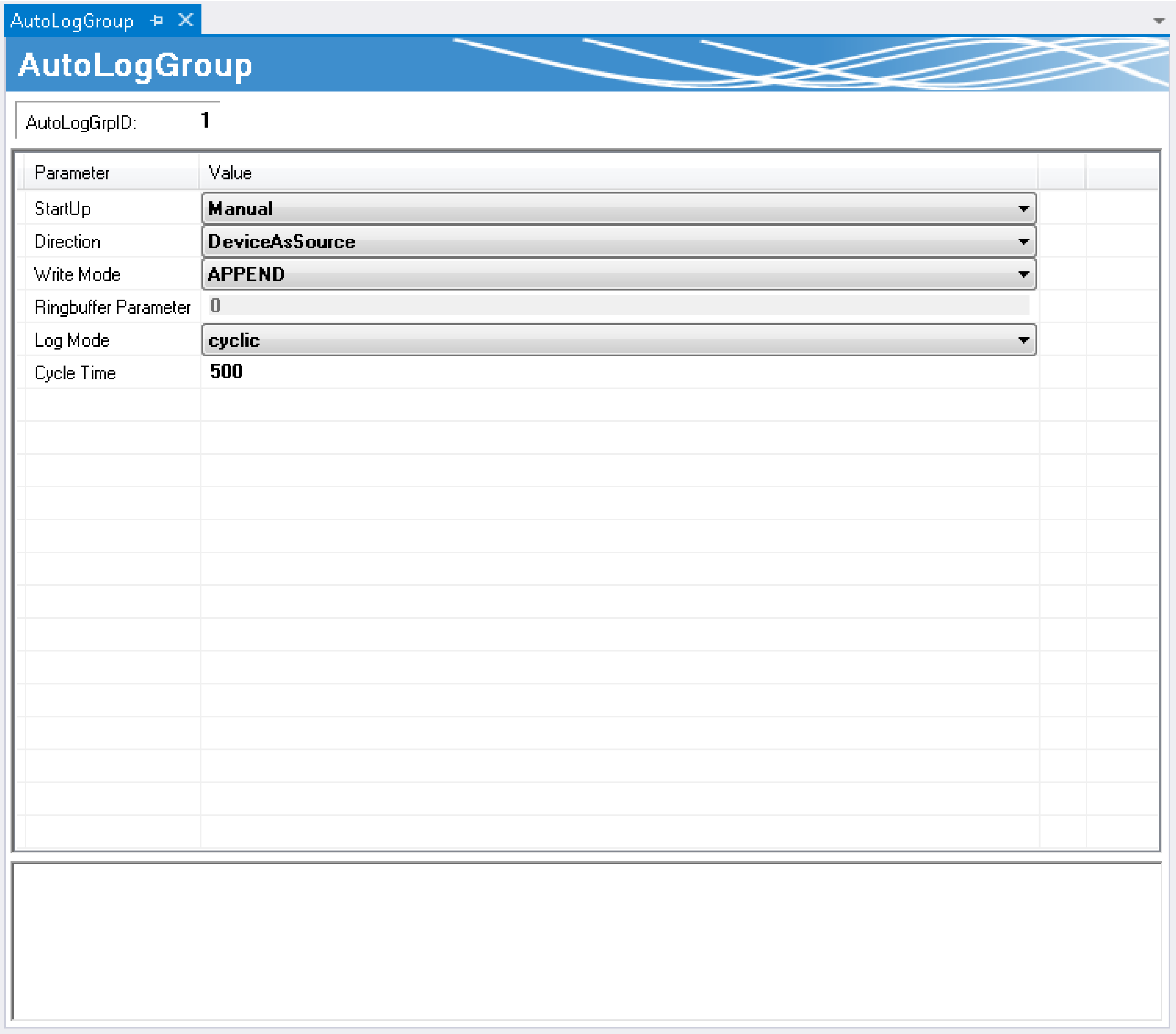
The ADS device is automatically created under an AutoLog group. In the most frequent use case the ADS device is the PLC runtime. The following parameters can be set in the editor:



ADSDevID: 1

Parameter	Value
ADS Device	local
AMS NetID	5.19.111.106.1.1
AMS Port	851
Timeout	2000
Connection Type	bySymbolName

ADS Device	Name of the ADS target device.
AMS NetID	Address of the target device in the TwinCAT network.
AMS Port	Port of the target device in the TwinCAT network.
Timeout	Time after which it is assumed that the connection to the target device is lost.
Connection Type	bySymbolName: Connection is established based on the symbol name. byIndexGroup: Connection is established based on the memory index.



AutoLogGrpID: 1

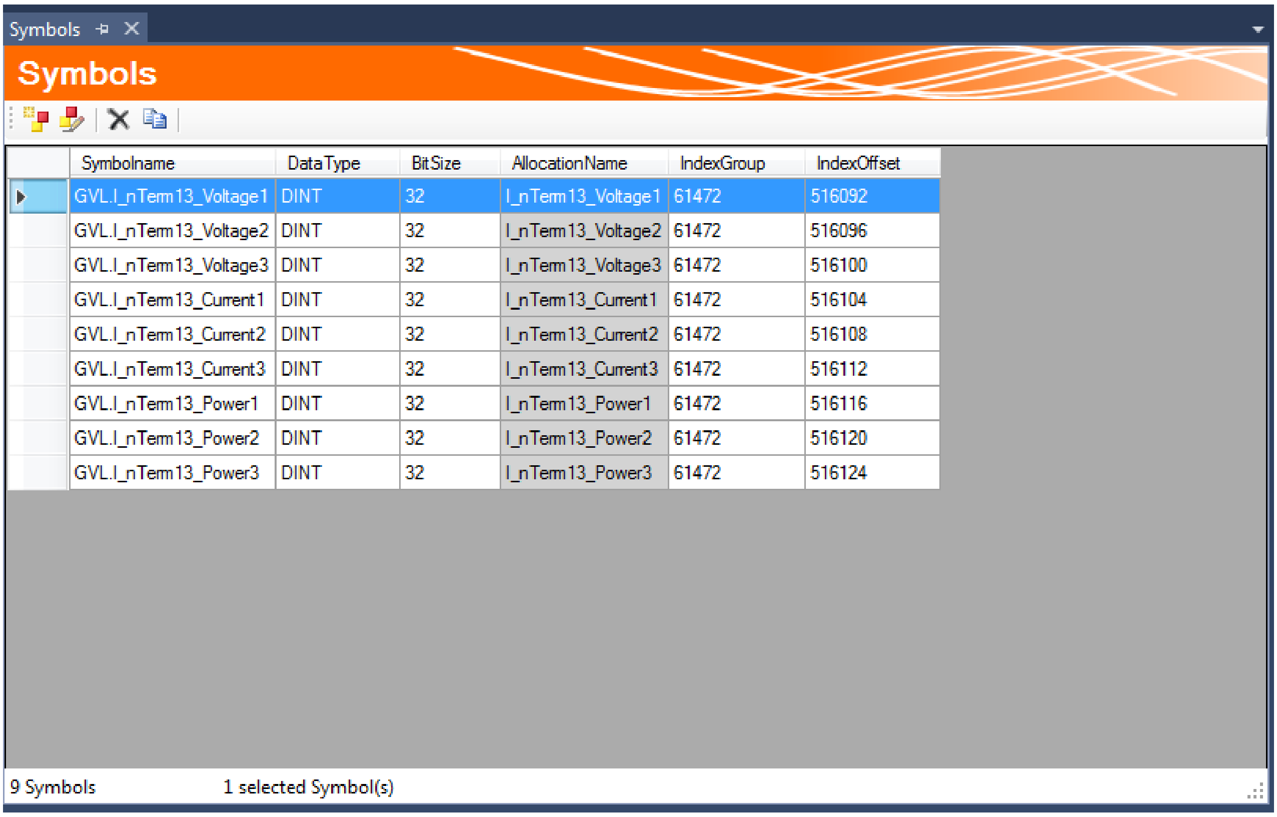
Parameter	Value
StartUp	Manual
Direction	DeviceAsSource
Write Mode	APPEND
Ringbuffer Parameter	0
Log Mode	cyclic
Cycle Time	500

StartUp	AutoLog mode can be enabled manually (with a command in the PLC or from the configurator) or automatically during system startup.
Direction	The set ADS device is used as data target or data source.
Write mode	The data can appended in a database line-by-line, held in a ring buffer on a temporal or quantitative basis, or simply be updated at the corresponding position.
Ring buffer parameter	Depending on the setting this parameter represent the time or the cycles after which the ring buffer is updated.
Log mode	The variable is written either after a certain cycle time or when a change occurs.
Cycle Time	Cycle time after which the variable is written.

A new AutoLog group for the database configuration can be added via the command **Add New AutologGroup** in the context menu of a database configuration or via the toolbar. These AutoLog groups refer to the parent database.

Configuring symbols

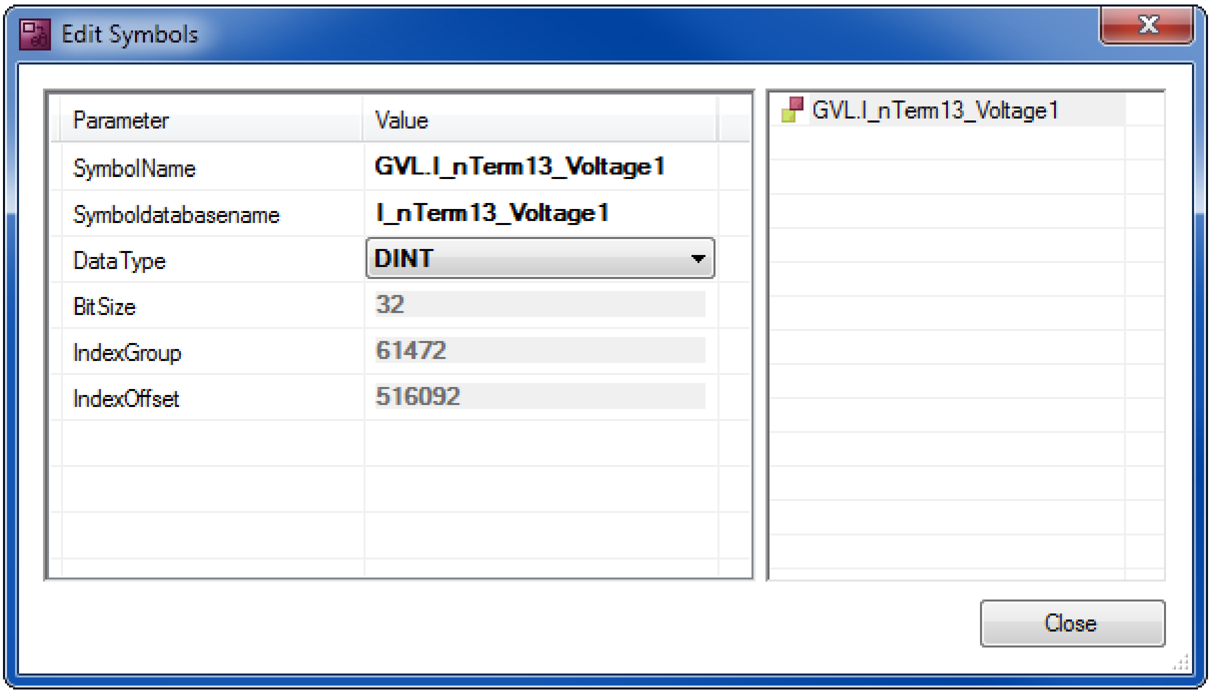
The symbols you set here are written to or read from the database, depending on whether the ADS device is the data target or the data source. The TwinCAT Target browser can be used for convenient access. Here you can search for the symbols on the target and communicate between the two tools via drag & drop.



	Symbolname	DataType	BitSize	AllocationName	IndexGroup	IndexOffset
▶	GVL.I_nTerm13_Voltage1	DINT	32	I_nTerm13_Voltage1	61472	516092
	GVL.I_nTerm13_Voltage2	DINT	32	I_nTerm13_Voltage2	61472	516096
	GVL.I_nTerm13_Voltage3	DINT	32	I_nTerm13_Voltage3	61472	516100
	GVL.I_nTerm13_Current1	DINT	32	I_nTerm13_Current1	61472	516104
	GVL.I_nTerm13_Current2	DINT	32	I_nTerm13_Current2	61472	516108
	GVL.I_nTerm13_Current3	DINT	32	I_nTerm13_Current3	61472	516112
	GVL.I_nTerm13_Power1	DINT	32	I_nTerm13_Power1	61472	516116
	GVL.I_nTerm13_Power2	DINT	32	I_nTerm13_Power2	61472	516120
	GVL.I_nTerm13_Power3	DINT	32	I_nTerm13_Power3	61472	516124

9 Symbols 1 selected Symbol(s)

Symbols can also be added manually to symbol groups or edited. The information that is required varies, depending on whether in the ADS device the connection type was selected via the symbol name or the index groups. The starting point is always the ADS device.



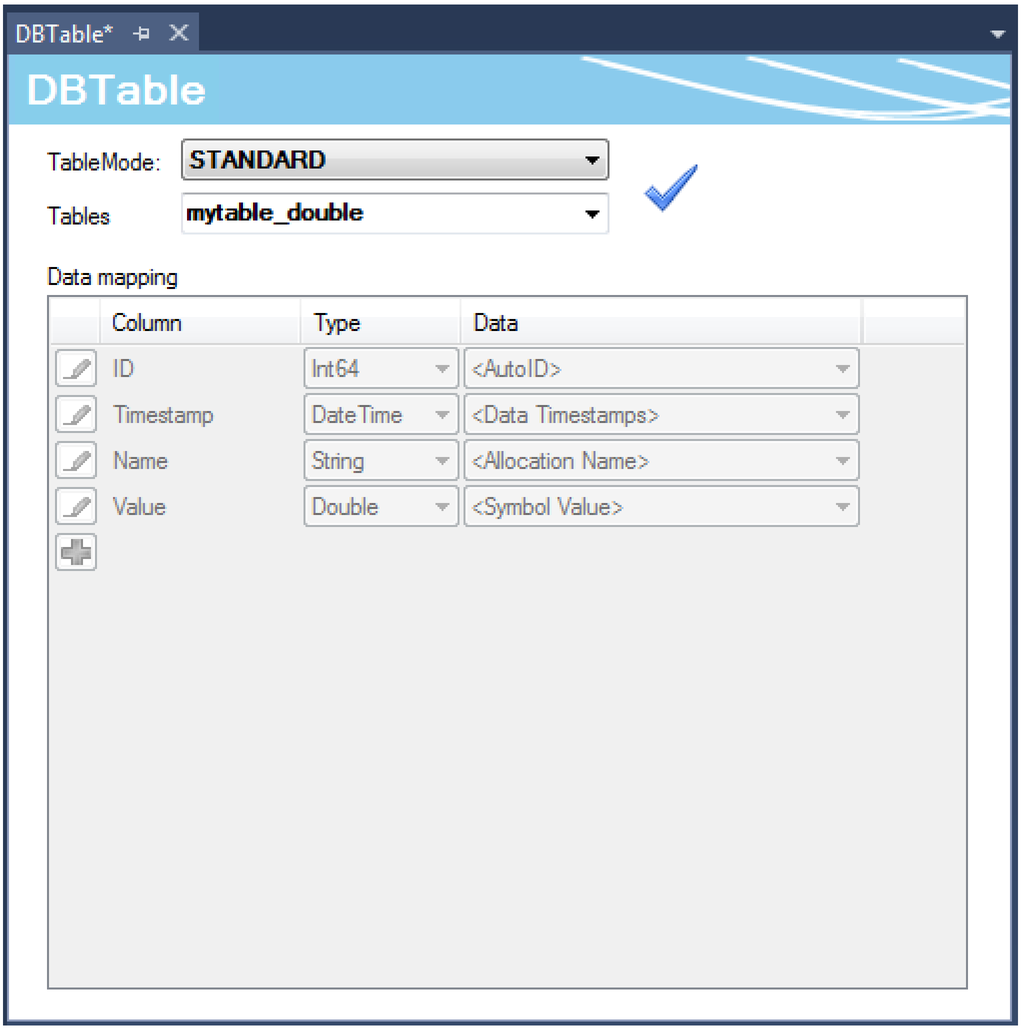
Parameter	Value
SymbolName	GVL.I_nTerm13_Voltage1
Symboldatabasename	I_nTerm13_Voltage1
DataType	DINT
BitSize	32
IndexGroup	61472
IndexOffset	516092

SymbolName	The symbol is addressed based on the set ADS device
Symbol database name	Name of the variable in the database table
DataType	PLC data type of the symbol
BitSize	Bit size of the symbols (set automatically for the data types)
IndexGroup	Index group in the TwinCAT system
IndexOffset	Index offset in the TwinCAT system

Configuring a table

The table in a database can be based on a standard table structure or on an individual structure.

The corresponding table can be selected from a list of possible tables. If the table does not yet exist, you can create it via the SQL Query Editor [▶ 108]. If you select the standard table structure, a blue tick indicates whether the selected table corresponds to this structure.



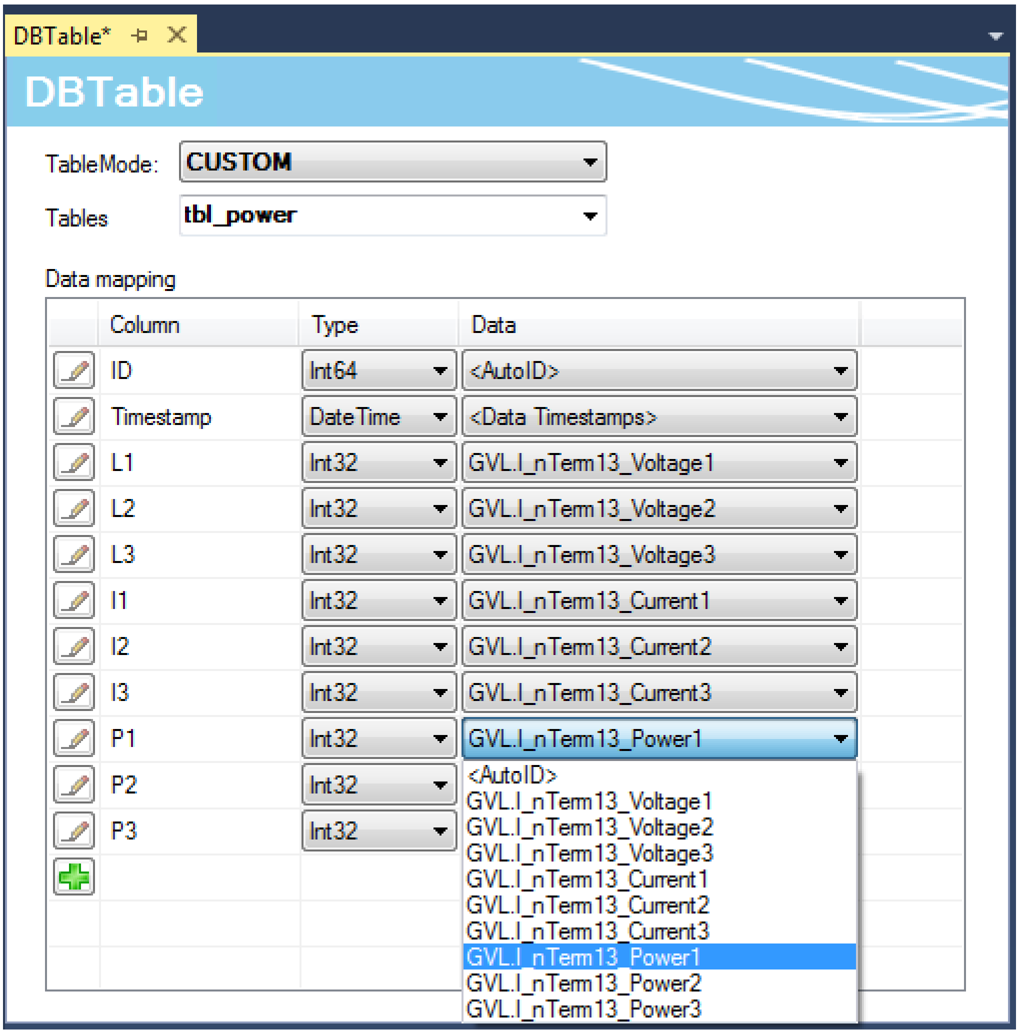
TableMode: STANDARD

Tables mytable_double ✓

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
Name	String	<Allocation Name>
Value	Double	<Symbol Value>

The specific table type offers the option to distribute the individual symbols that were set in the symbol group to the table columns in the database as required. When a record is written to the database in AutoLog mode, the current values of the symbol group at the sampling time are saved in the corresponding table column.



TableMode: CUSTOM

Tables: tbl_power

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
L1	Int32	GVL.I_nTerm13_Voltage1
L2	Int32	GVL.I_nTerm13_Voltage2
L3	Int32	GVL.I_nTerm13_Voltage3
I1	Int32	GVL.I_nTerm13_Current1
I2	Int32	GVL.I_nTerm13_Current2
I3	Int32	GVL.I_nTerm13_Current3
P1	Int32	GVL.I_nTerm13_Power1
P2	Int32	
P3	Int32	

Write direction mode

The TwinCAT Database Server has four different write direction modes. These are explained below.

DB_TO_ADS

This write mode is used to cyclically read variable values from a database and write the read values into PLC variables.

ADS_TO_DB_APPEND

This write mode is used to cyclically write variable values from the PLC into a database. Each time a new record is created and appended at the end of the table/file.

ADS_TO_DB_UPDATE

This write mode is used to cyclically read variable values from the PLC and compare the read values with the database records. If differences are detected, the corresponding record is modified with the new value.

ADS_TO_DB_RINGBUFFER

This write mode can be used to specify the number of records or the age of records.
This write mode is available during cyclic logging via the symbol groups and during logging with the function block FB_DBWrite.
The RingBuffer mode is available for all database types. This mode can also be used to influence logging in ASCII files.

RingBuffer-Arten

The RingBuffer can be used in two different ways:

- "RingBuffer_Time"
- "RingBuffer_Count"

RingBuffer Time

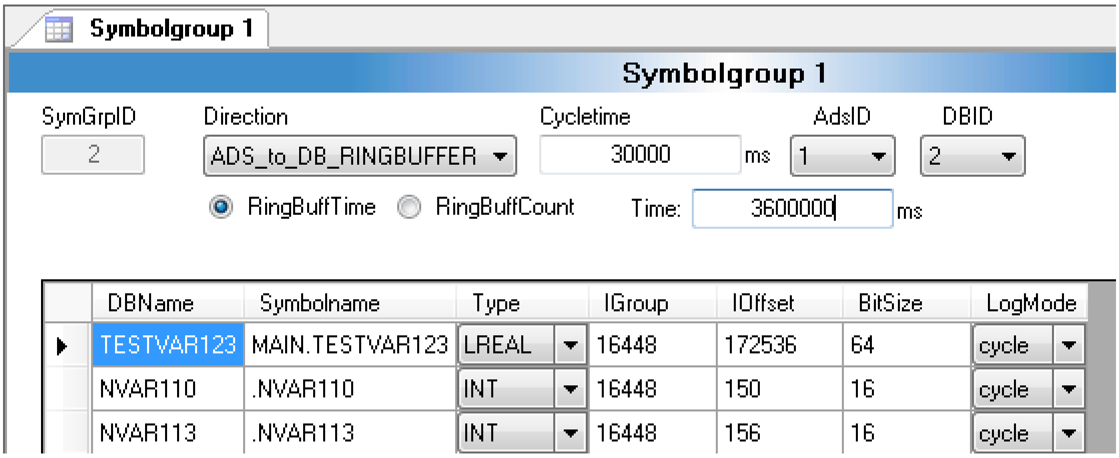
In this mode a time can be specified for the maximum age of the record. If this age is exceeded, the corresponding record is deleted.

RingBuffer Count

In this mode a maximum number of records can be specified. When the maximum number is reached, the oldest records are deleted in order to make room for the new ones.

Declaring the RingBuffer mode in the XML configuration file editor

RingBuffer_Time:



Symbolgroup 1

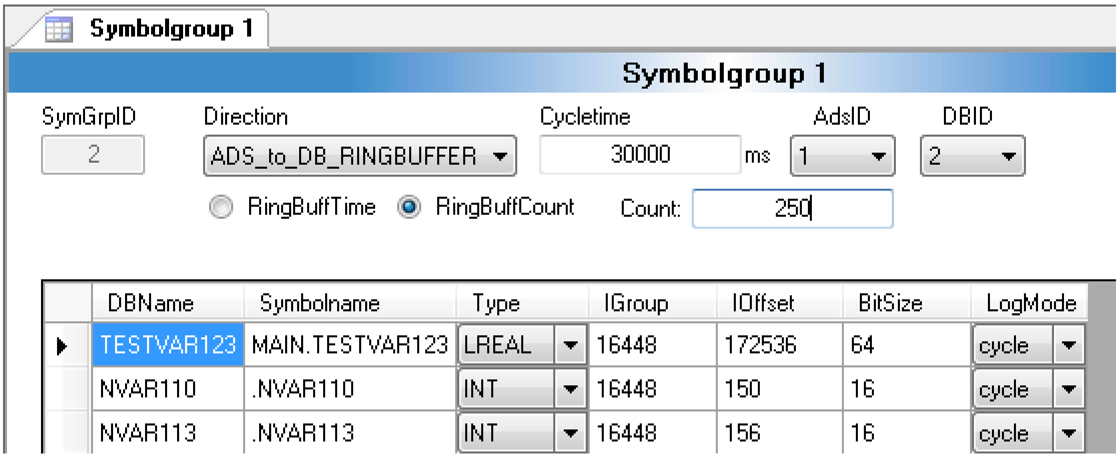
SymGrpID: 2
Direction: ADS_to_DB_RINGBUFFER
Cycletime: 30000 ms
AdsID: 1
DBID: 2

● RingBuffTime ○ RingBuffCount Time: 3600000 ms

	DBName	Symbolname	Type	IGroup	IOffset	BitSize	LogMode
▶	TESTVAR123	MAIN.TESTVAR123	LREAL	16448	172536	64	cycle
	NVAR110	.NVAR110	INT	16448	150	16	cycle
	NVAR113	.NVAR113	INT	16448	156	16	cycle

The time is specified in milliseconds.

RingBuffer_Count:



Symbolgroup 1

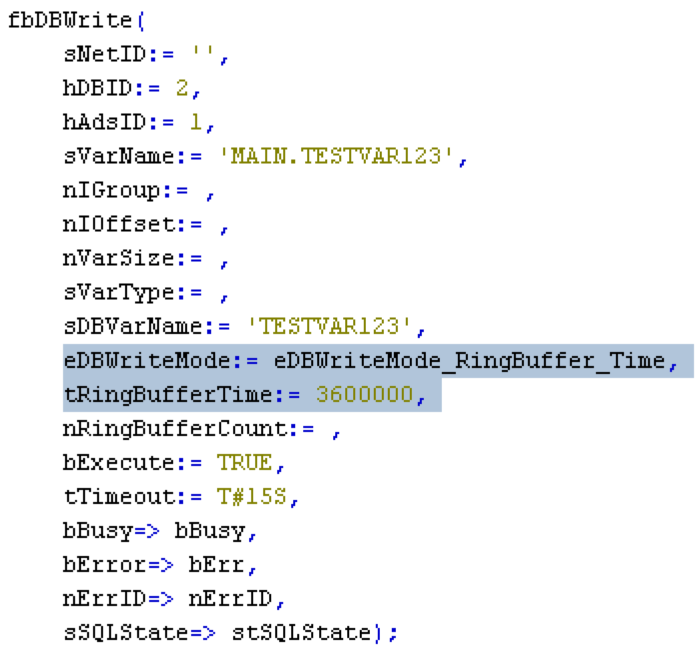
SymGrpID: 2
Direction: ADS_to_DB_RINGBUFFER
Cycletime: 30000 ms
AdsID: 1
DBID: 2

○ RingBuffTime ● RingBuffCount Count: 250

	DBName	Symbolname	Type	IGroup	IOffset	BitSize	LogMode
▶	TESTVAR123	MAIN.TESTVAR123	LREAL	16448	172536	64	cycle
	NVAR110	.NVAR110	INT	16448	150	16	cycle
	NVAR113	.NVAR113	INT	16448	156	16	cycle

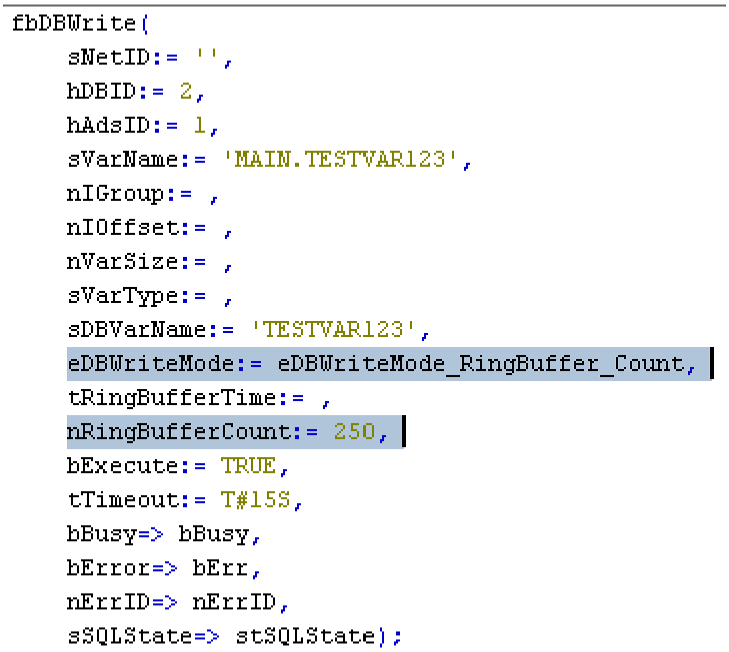
Declaring the RingBuffer mode in FB_DBWrite:

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);
```

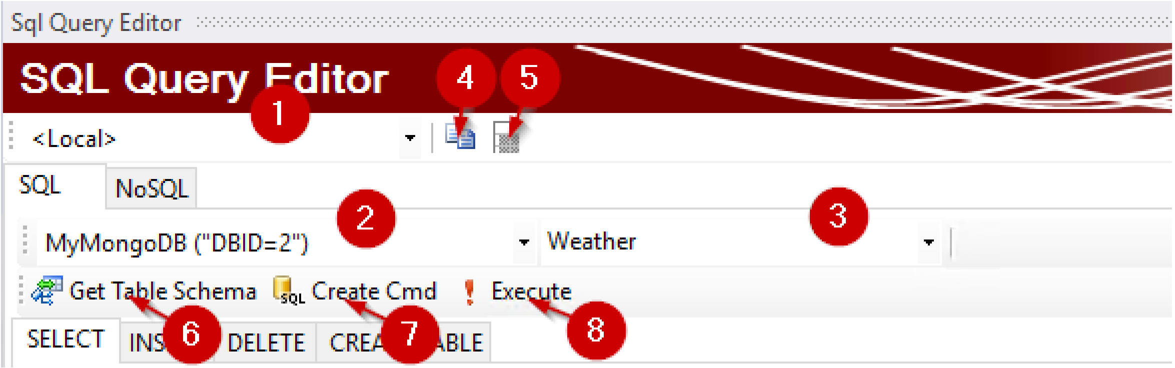
RingBuffer_Count:



```
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

The SQL Query Editor is a Database Server tool that supports the development of your application. The tool can be used to test connections and SQL commands and to check the compatibility between PLC and databases.



ID	Name	Function
1	Target system	Choose Target System with installed TwinCAT Database Server
2	Database	Selecting the configured database connection
3	Table	Selecting the existing tables in the database
4	Copying for PLC	Copying the SQL command to the PLC string. This can be copied into the PLC source code. Special characters are automatically captured and formatted.
5	Export TC3	Exporting the table schema into a PLC structure. This can be used in the program for SQL commands, for example.
6	Get Table Schema	Reading the table structure
7	Create Cmd	Creating an SQL command, based on the table structure
8	Execute	Executing the SQL command

First select the target system from the routes of your TwinCAT system (1). The TwinCAT Database Server must be installed on the target system. If a NoSQL database is stored in the configuration, an additional NoSQL tab is visible. You will find the documentation in a subitem below.

All configured databases (2) are displayed, once you have activated the database configurations on the target system. You can also select one of the available tables (3) from the database. Based on this table, you can generate SQL commands from the SQL Query Editor and send them to the database. The SQL commands have different syntax, depending on database type.

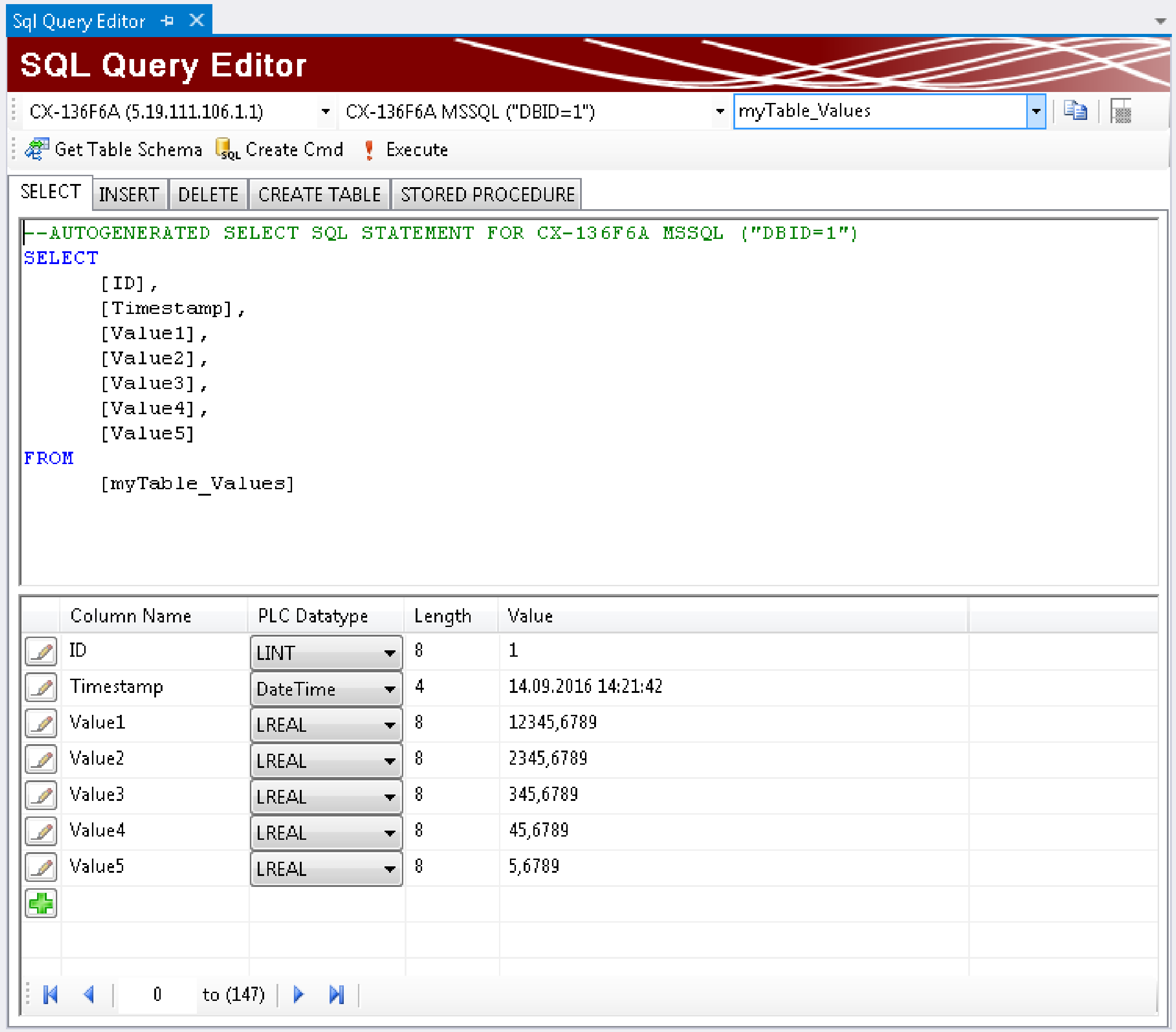
Three commands are available for generating the individual SQL commands:

- Get Table Schema: Calls up the structure of the selected table.
    - Information such as the column name, PLC data type and size of variables is displayed. The retrieved structure can also be prepared for your PLC application via the commands **Copy for PLC** (4) or **Export TC3** (5).
- Create Cmd: An SQL command is generated in the command text box, depending on the selected tab. The command syntax may differ, depending on the database type. The previously read table schema is used here.
    - The created SQL command can optionally be modified.
- Execute: The SQL command shown in the text box is executed and returns values, if applicable.

The differences in the individual SQL commands are explained below.

Select command

Select commands can be created and sent via the **SELECT** tab. Select commands give the opportunity to read records from the databases. After executing the command, values are returned if they exist in the table. They are listed under "Value" in the table structure display. Use the arrows under the display to navigate through the individual records.



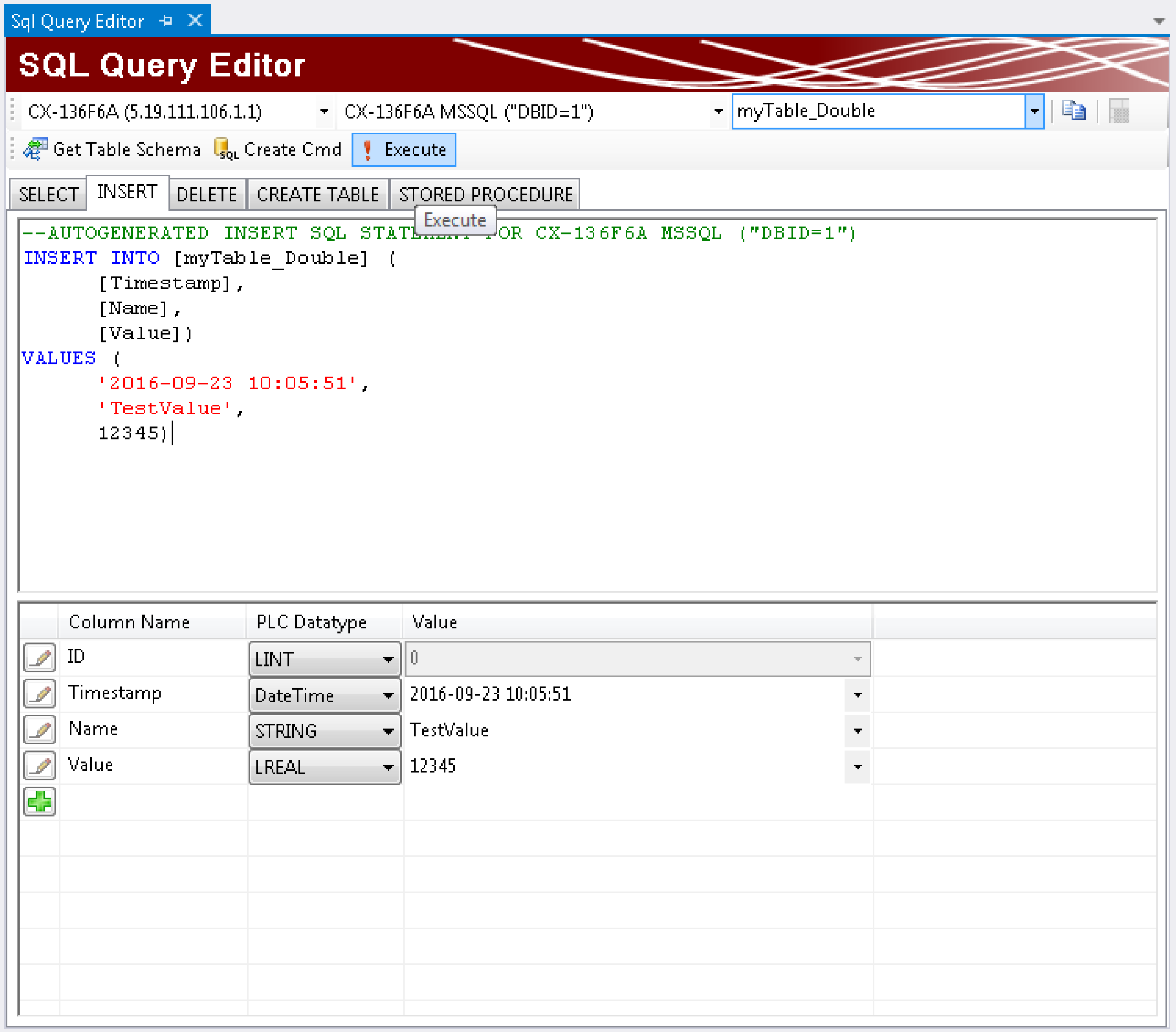
```
--AUTOGENERATED SELECT SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
SELECT
        [ID],
        [Timestamp],
        [Value1],
        [Value2],
        [Value3],
        [Value4],
        [Value5]
FROM
        [myTable_Values]
```

Column Name	PLC Datatype	Length	Value
ID	LINT	8	1
Timestamp	DateTime	4	14.09.2016 14:21:42
Value1	LREAL	8	12345,6789
Value2	LREAL	8	2345,6789
Value3	LREAL	8	345,6789
Value4	LREAL	8	45,6789
Value5	LREAL	8	5,6789

Insert command

The Insert command gives the opportunity to write records into the table. The values under "Value" can be modified once the table structure has been retrieved. If the command is then generated, the values in the Insert command will automatically be in the right format. These values are written into the table when the command is executed.

This value cannot be customized if automatic ID generation is used.



```
--AUTOGENERATED INSERT SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
INSERT INTO [myTable_Double] (
       [Timestamp],
       [Name],
       [Value])
VALUES (
       '2016-09-23 10:05:51',
       'TestValue',
       12345)
```

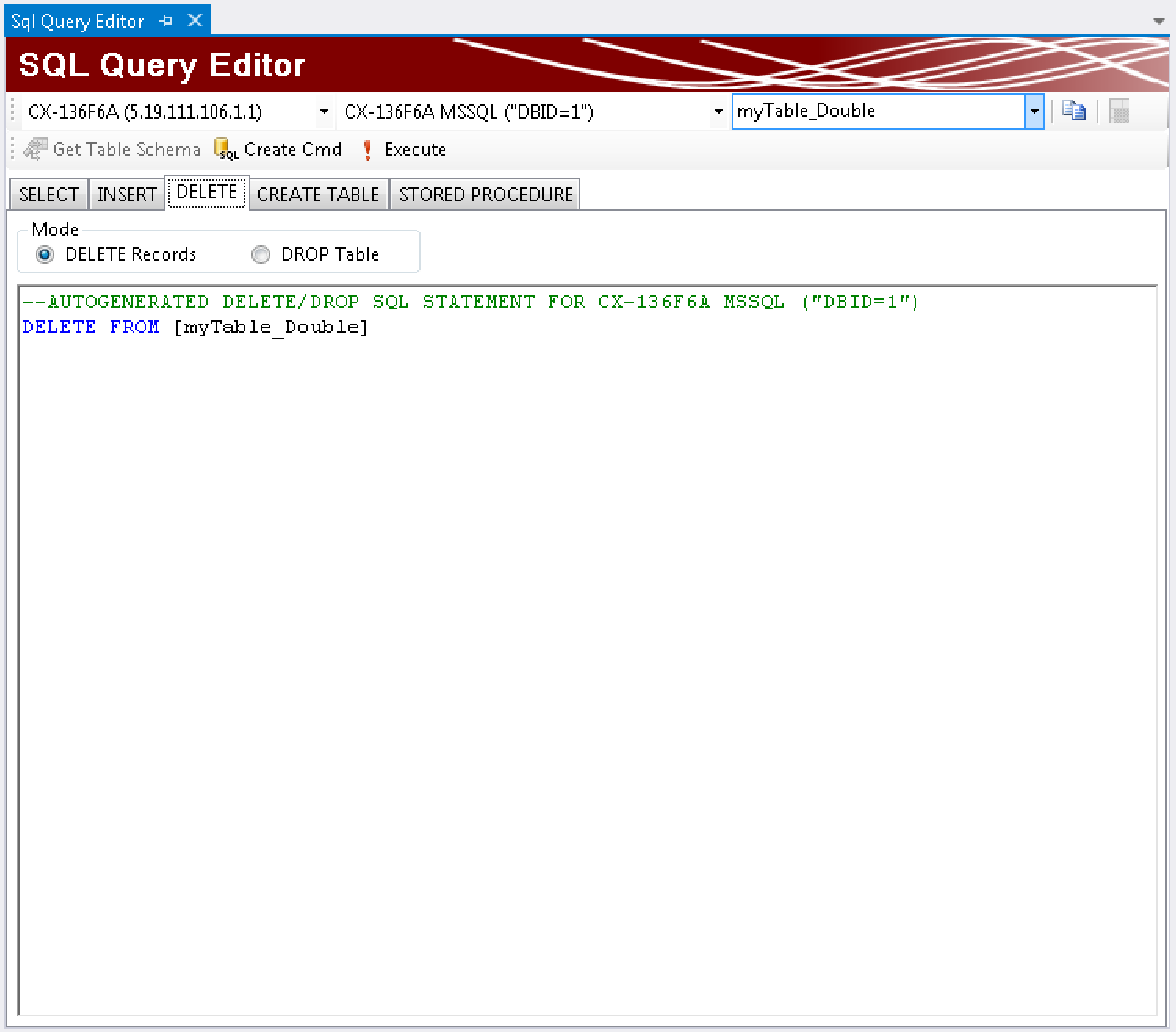
Column Name	PLC Datatype	Value
ID	LINT	0
Timestamp	DateTime	2016-09-23 10:05:51
Name	STRING	TestValue
Value	LREAL	12345

Delete command

The Delete command has two functions.

1. **DELETE Records**: Deletes the contents of a table.
2. **DROP table:** Deletes the whole table.

This SQL command can also be customized, in order to delete only a particular section of the table, for example.



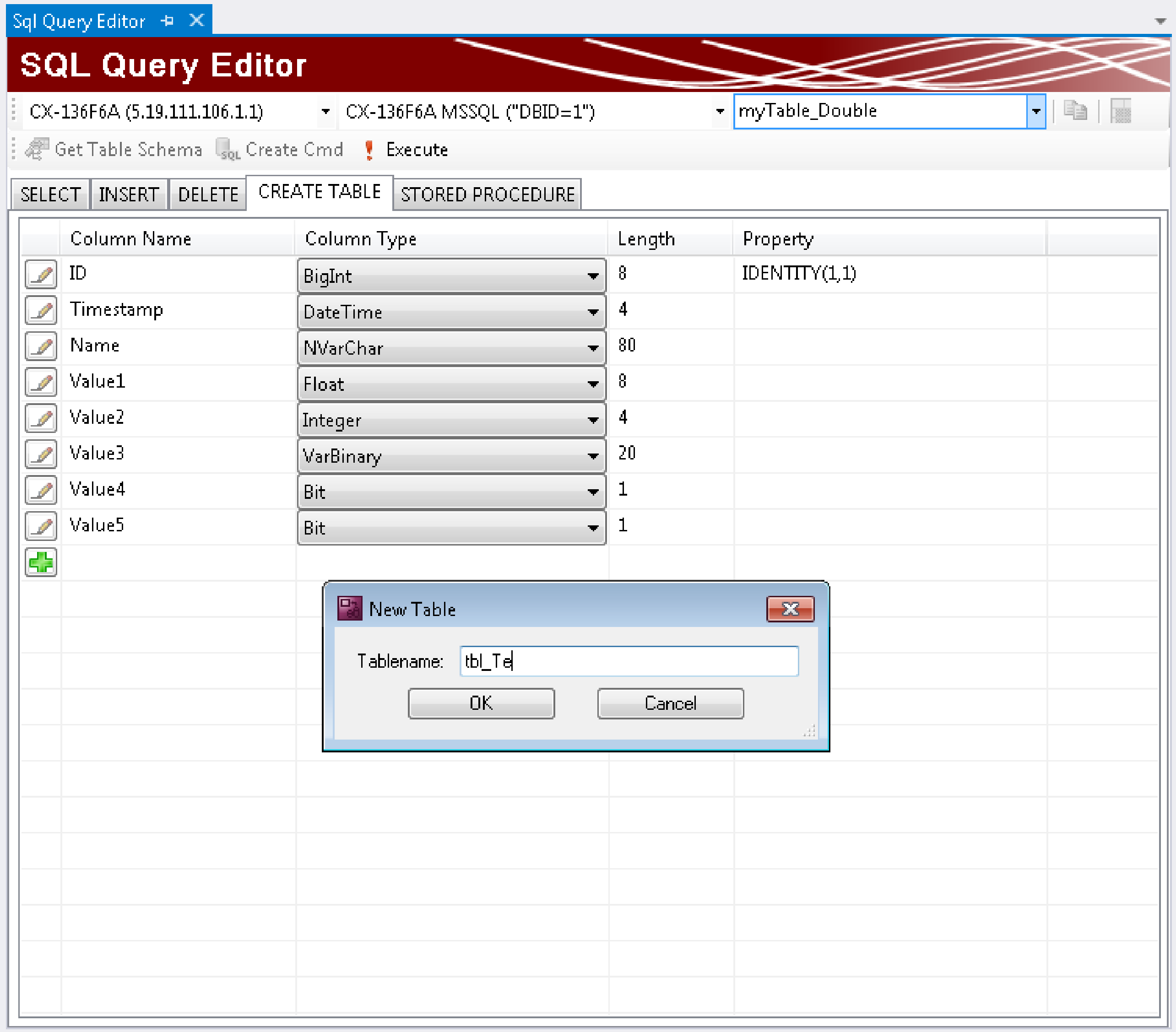
● DELETE Records ○ DROP Table

```
--AUTOGENERATED DELETE/DROP SQL STATEMENT FOR CX-136F6A MSSQL ("DBID=1")
DELETE FROM [myTable_Double]
```

Create Table command

The **CREATE TABLE** tab can be used to create tables within the database. Further columns can be added to the table with (+), as required. Once you have specified the column name and type, you can specify additional properties, in order to generate automatic IDs, for example.

The table name can be determined by executing the command. The table with the configured table structure is created.



CX-136F6A (5.19.111.106.1.1)

CX-136F6A MSSQL ("DBID=1")

myTable_Double

Column Name	Column Type	Length	Property
ID	BigInt	8	IDENTITY(1,1)
Timestamp	DateTime	4	
Name	NVarChar	80	
Value1	Float	8	
Value2	Integer	4	
Value3	VarBinary	20	
Value4	Bit	1	
Value5	Bit	1	

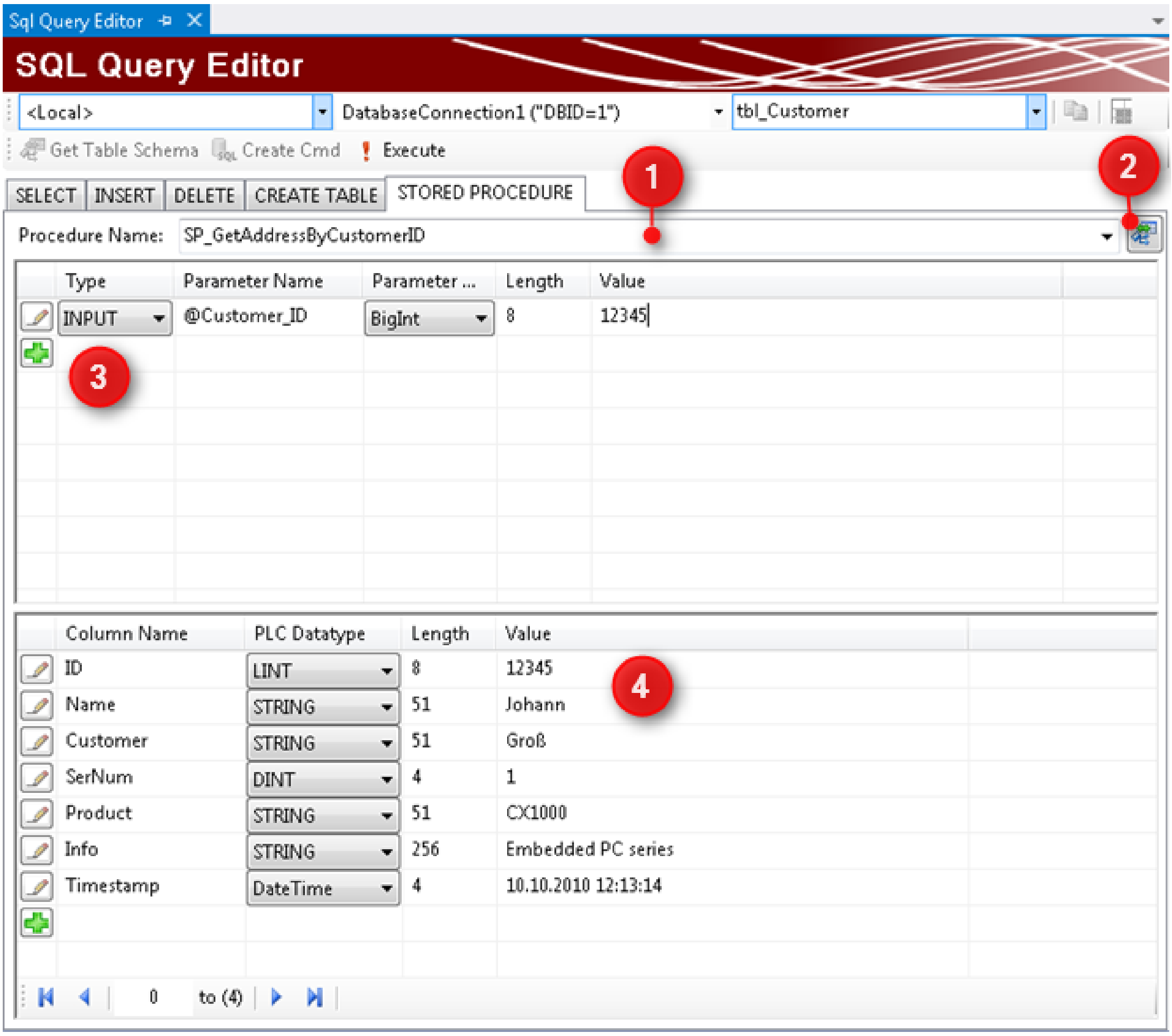
Tablename: tbl_Te

Stored Procedures

The TwinCAT Database Server supports "Stored Procedures", which provide numerous databases for processing more complex queries at the database level or to make a simplified interface available.

If Stored Procedures are available in the database and the table, you can list and select them (1). The input and output parameters can be picked up automatically (2) and transferred to the tables in the display (3)(4).

The parameter type, name and data type are displayed there. In addition you can insert values here, in order to execute the Stored Procedures with the input values via "Execute". The result is displayed in the output values (4). If several records are returned, the arrow keys can be used to switch between them. This functionality serves as development aid for the call in the PLC. The results are returned there by calling the corresponding function block [▶ 191].



Procedure Name: SP_GetAddressByCustomerID

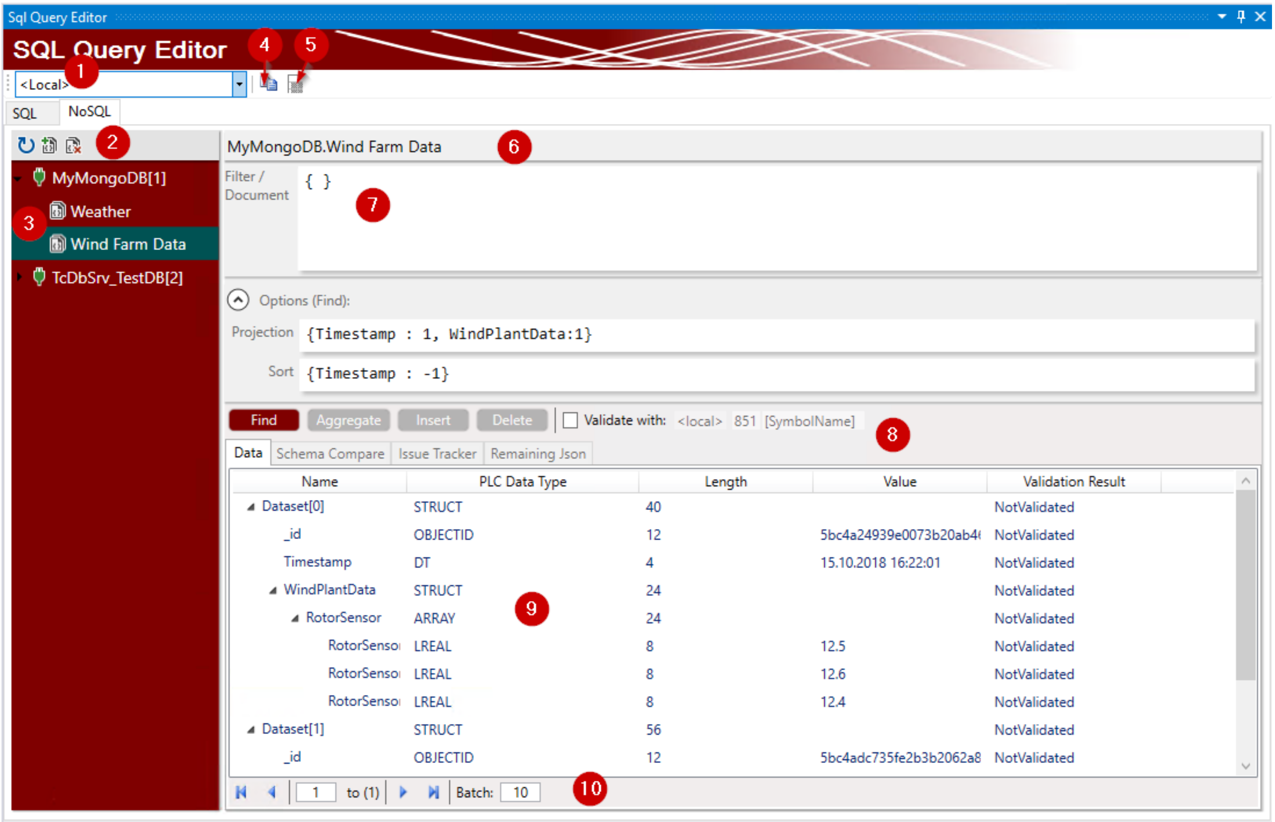
Type	Parameter Name	Parameter ...	Length	Value
INPUT	@Customer_ID	BigInt	8	12345

Column Name	PLC Datatype	Length	Value
ID	LINT	8	12345
Name	STRING	51	Johann
Customer	STRING	51	Groß
SerNum	DINT	4	1
Product	STRING	51	CX1000
Info	STRING	256	Embedded PC series
Timestamp	DateTime	4	10.10.2010 12:13:14

NoSql Query Editor

The NoSQL tab supports the special functions of NoSQL databases. It is only visible if a NoSQL database has been configured and uploaded to your target device (1).

Your available databases are then listed in the list of NoSQL databases (3). SQL databases are not displayed in this list. All existing collections are now listed under a selected database. In the menu bar (2) the list can be updated, and collections can be added or deleted.



<Local>

MyMongoDB.Wind Farm Data

Filter / Document: { }

Options (Find):

Projection: {Timestamp : 1, WindPlantData:1}

Sort: {Timestamp : -1}

☐ Validate with: <local> 851 [SymbolName]

Name	PLC Data Type	Length	Value	Validation Result
Dataset[0]	STRUCT	40		NotValidated
_id	OBJECTID	12	5bc4a24939e0073b20ab4(	NotValidated
Timestamp	DT	4	15.10.2018 16:22:01	NotValidated
WindPlantData	STRUCT	24		NotValidated
RotorSensor	ARRAY	24		NotValidated
RotorSenso	LREAL	8	12.5	NotValidated
RotorSenso	LREAL	8	12.6	NotValidated
RotorSenso	LREAL	8	12.4	NotValidated
Dataset[1]	STRUCT	56		NotValidated
_id	OBJECTID	12	5bc4adc735fe2b3b2062a8	NotValidated

1 to (1) Batch: 10

ID	Name	Function
1	Target system	Selecting the target system with installed TwinCAT Database Server
2	Database menu	Updating, adding and deleting collections
3	Database explorer	Selecting the existing database and collections
4	Copying for PLC	Copying the SQL command to a PLC string. This can be copied into the PLC source code. Special characters are automatically captured and formatted.
5	Export TC3	Exporting the table schema into a PLC structure. This can be used in the program for SQL commands, for example.
6	Collection path	Database name and collection that is currently selected.
7	Document/Filter	Depending on which function is used, this input field acts as a document or as a filter in JSON format. If you want to execute a Find operation and also carry out a projection or sort operation, you can fill these fields with *Options(Find)* below.
8	Control elements	Control elements for interaction with the TwinCAT Database Server.
9	Data display	List of returned data
10	Navigation	Allows iteration through the returned records

Once a collection has been selected, the target (6) of the query to be sent changes. The following functions (8) are provided:

Find: Executes a search query with the filter entered in the text field (7). Optionally, a projection or sorting operation can also be executed via the *Options(Find)* fields. Data is returned and listed in the data display (9). The syntax of the filters is database-specific.

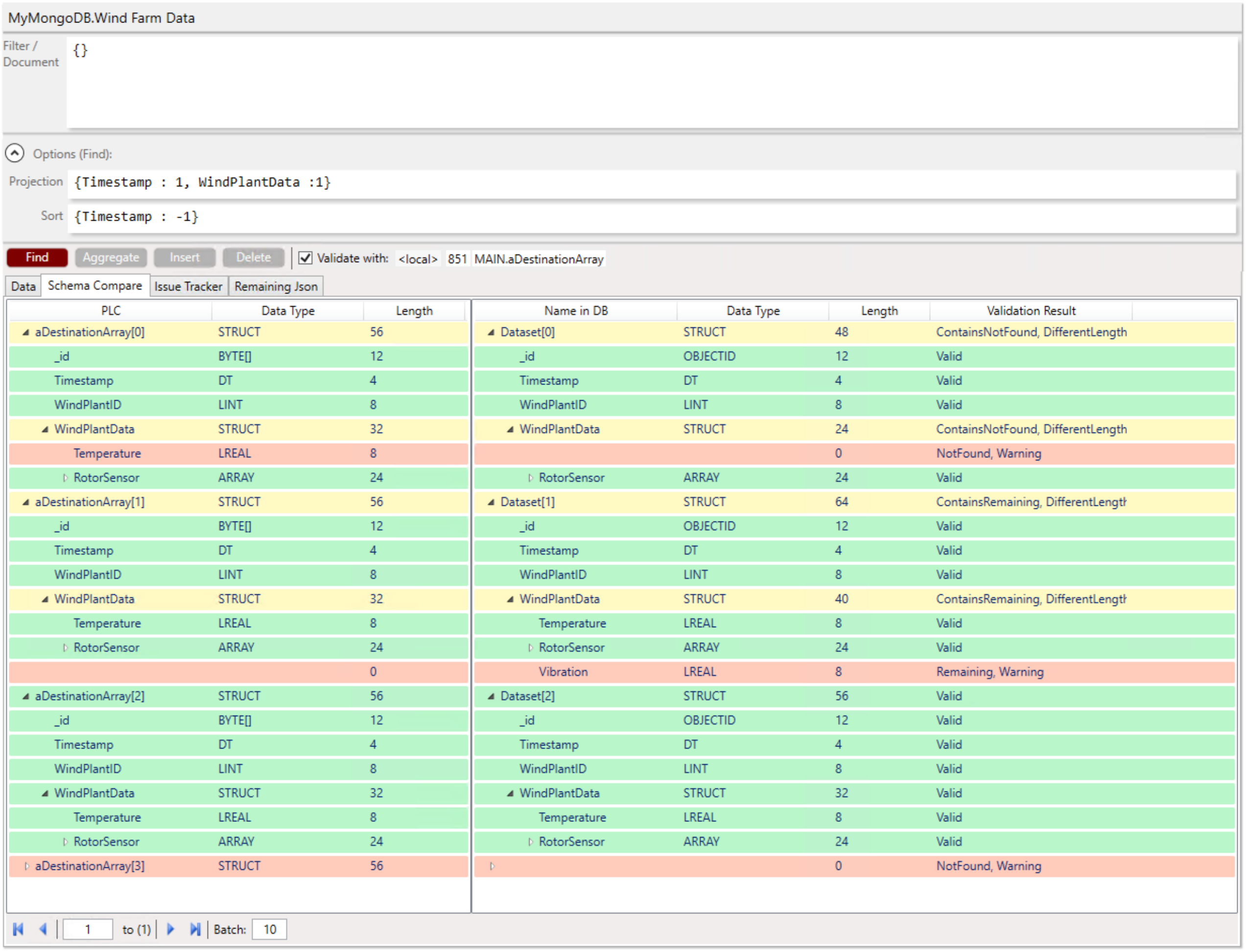
Aggregate: Executes an aggregation with the parameters entered in the text field (7). Data is returned and listed in the data display (9). The syntax of the filters is database-specific.

Insert: Executes an insert query of the (JSON) document or document array entered in the text field (7). These are then written to the collection.

Delete: Executes a delete query on the data found with the filter in the text field (7). Any data that is found is deleted from the collection.

Validate: If this option is selected, the data queries are not automatically parsed according to their own schema, but an attempt is made to map these data to the structure of the symbol from the PLC, which was specified via these parameters.

With the latter function, a Find query may lead to conflicts. In contrast to structures in the PLC process image, records in NoSQL databases do not have to follow a fixed schema. It is therefore possible that queried documents have no data relating to a specific element in the PLC structure. Or the record carries data that does not occur in the PLC structure. These data are assigned via the name or the attribute "ElementName" in the PLC.



MyMongoDB.Wind Farm Data

Filter / Document: {}

Options (Find):

Projection: {Timestamp : 1, WindPlantData :1}

Sort: {Timestamp : -1}

☑ Validate with: <local> 851 MAIN.aDestinationArray

PLC	Data Type	Length	Name in DB	Data Type	Length	Validation Result
aDestinationArray[0]	STRUCT	56	Dataset[0]	STRUCT	48	ContainsNotFound, DifferentLength
_id	BYTE[]	12	_id	OBJECTID	12	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
WindPlantID	LINT	8	WindPlantID	LINT	8	Valid
WindPlantData	STRUCT	32	WindPlantData	STRUCT	24	ContainsNotFound, DifferentLength
Temperature	LREAL	8			0	NotFound, Warning
RotorSensor	ARRAY	24	RotorSensor	ARRAY	24	Valid
aDestinationArray[1]	STRUCT	56	Dataset[1]	STRUCT	64	ContainsRemaining, DifferentLength
_id	BYTE[]	12	_id	OBJECTID	12	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
WindPlantID	LINT	8	WindPlantID	LINT	8	Valid
WindPlantData	STRUCT	32	WindPlantData	STRUCT	40	ContainsRemaining, DifferentLength
Temperature	LREAL	8	Temperature	LREAL	8	Valid
RotorSensor	ARRAY	24	RotorSensor	ARRAY	24	Valid
		0	Vibration	LREAL	8	Remaining, Warning
aDestinationArray[2]	STRUCT	56	Dataset[2]	STRUCT	56	Valid
_id	BYTE[]	12	_id	OBJECTID	12	Valid
Timestamp	DT	4	Timestamp	DT	4	Valid
WindPlantID	LINT	8	WindPlantID	LINT	8	Valid
WindPlantData	STRUCT	32	WindPlantData	STRUCT	32	Valid
Temperature	LREAL	8	Temperature	LREAL	8	Valid
RotorSensor	ARRAY	24	RotorSensor	ARRAY	24	Valid
aDestinationArray[3]	STRUCT	56			0	NotFound, Warning

The differences in the data can be examined via the "Schema Compare" tab. The above sample shows that the variable "*Temperature"* of data type *LREAL* has been created in the PLC structure "*WindPlantData*" for the first document returned. However, the read record has no data for this variable. In the second document the variable *"Vibration"* is missing in the PLC. The corresponding colors show the weighting of the conflict:

Red: too many or too few data available.
Yellow: The byte length of the record does not match, or underlying records are left over or missing.
Green: No conflicts

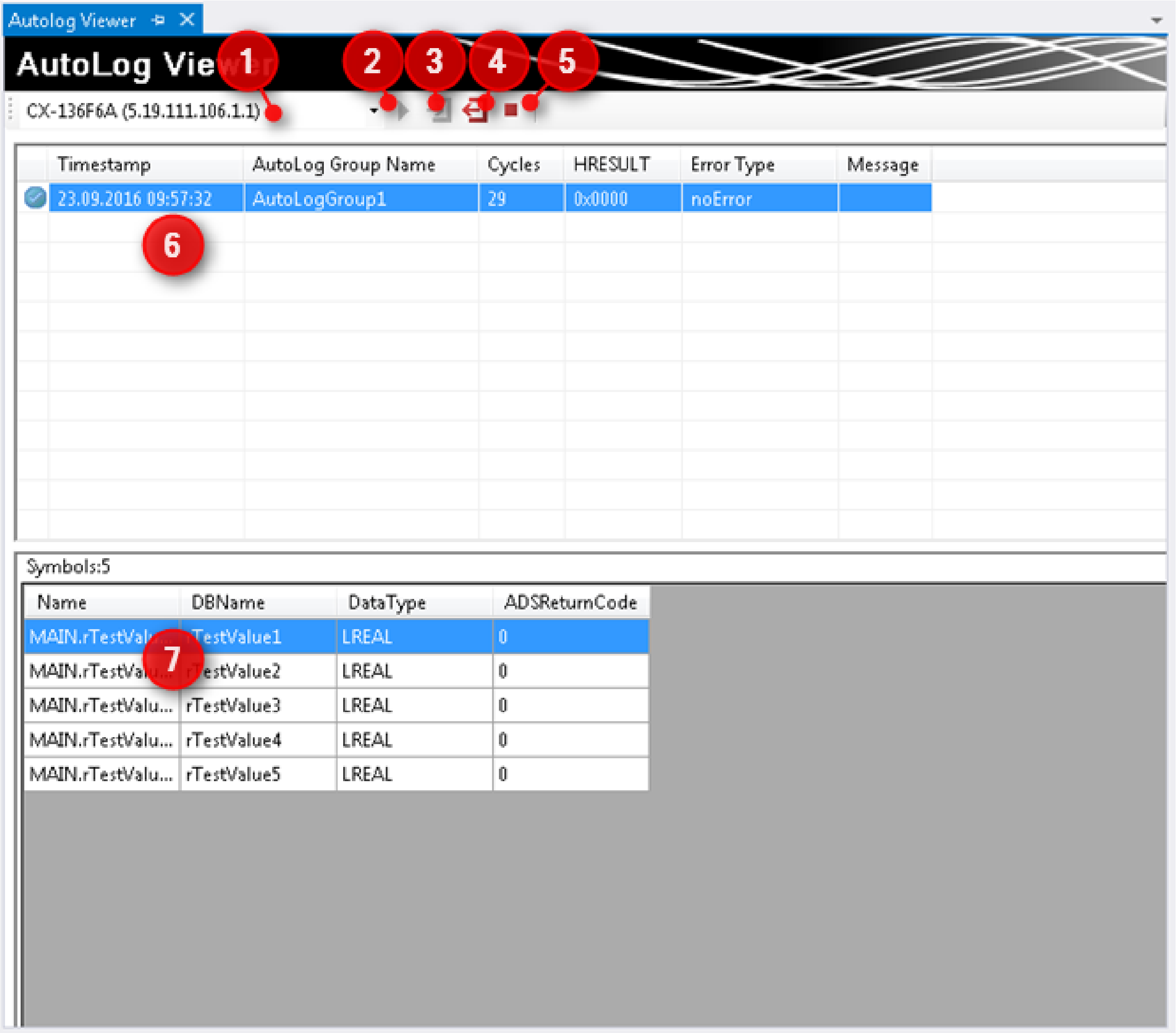
These conflicts are also listed under the "Issue Tracker" tab. It can also be read into the PLC as a string array, if required.

The "Remaining JSON" tab returns any remaining records as JSON. This information can also be read into the PLC as a string.

The control elements in the status bar can be used to iterate through the data, similar to the SQL tab. The number of records displayed simultaneously can be specified.

5.1.2.1.6 AutoLog Live View

The AutoLog Viewer of the TwinCAT Database Server is a tool for controlling and monitoring the AutoLog mode. You can log into a target system, similar to the TwinCAT PLC. In logged-in state the AutoLog mode can be started or stopped. Information on the current state of the logging is shown in the lower part of the window. When an AutoLog group is selected, further information is displayed via the logged symbols.



CX-136F6A (5.19.111.106.1.1)

Timestamp	AutoLog Group Name	Cycles	HRESULT	Error Type	Message
23.09.2016 09:57:32	AutoLogGroup1	29	0x0000	noError	

Symbols:5

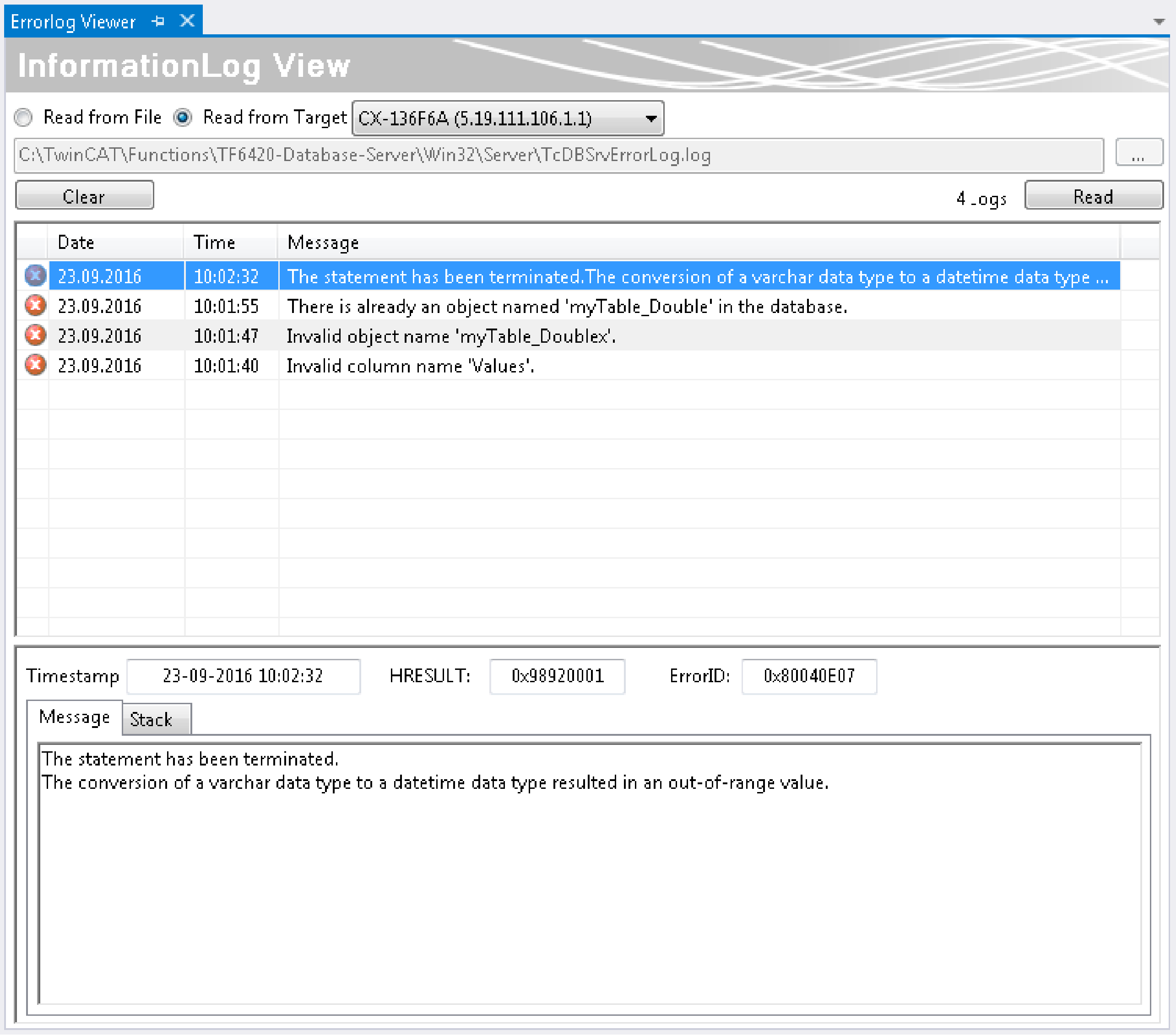
Name	DBName	DataType	ADSReturnCode
MAIN.rTestVal	TestValue1	LREAL	0
MAIN.rTestVa	rTestValue2	LREAL	0
MAIN.rTestValu...	rTestValue3	LREAL	0
MAIN.rTestValu...	rTestValue4	LREAL	0
MAIN.rTestValu...	rTestValue5	LREAL	0

ID	Name	Function
1	Target system	Choose Target System with installed TwinCAT Database Server
2	Start	Manual start of the AutoLog mode
3	Login	Logging into the active AutoLog process
4	Logout	Logging out of the active AutoLog process
5	Stop	Manual stop of the AutoLog mode
6	AutoLog groups	List of configured AutoLog groups on the target system
7	Symbols	List of configured symbols for the selected AutoLog group

5.1.2.1.7 InformationLog View

InformationLog View is a tool for reading log files from the TwinCAT Database Server. Recorded information is displayed with a timestamp, IDs and error messages in plain text.

The log files can not only be viewed or emptied via direct file access, but also directly via the target. This is particularly advantageous with distributed Database Servers in a network, for quick and easy access to the log file. For this access a route to the target device must exist.



○ Read from File ● Read from Target CX-136F6A (5.19.111.106.1.1)

C:\TwinCAT\Functions\TF6420-Database-Server\Win32\Server\TcDBSrvErrorLog.log

4 Logs

Date	Time	Message
23.09.2016	10:02:32	The statement has been terminated.The conversion of a varchar data type to a datetime data type ...
23.09.2016	10:01:55	There is already an object named 'myTable_Double' in the database.
23.09.2016	10:01:47	Invalid object name 'myTable_Doublex'.
23.09.2016	10:01:40	Invalid column name 'Values'.

Timestamp 23-09-2016 10:02:32 HRESULT: 0x98920001 ErrorID: 0x80040E07

The statement has been terminated.
The conversion of a varchar data type to a datetime data type resulted in an out-of-range value.

5.2 Databases

The TwinCAT Database Server is the link between TwinCAT PLC controllers and database systems. It supports a wide range of databases. In addition to conventional databases such as Microsoft SQL or Oracle, XML and ASCII files can also be used as databases. With ODBC databases it is even possible to enter database connection strings for communication with databases that are not listed as supported database types.

The configuration of the individual databases and mapping of the records in the PLC is explained on the following sections.

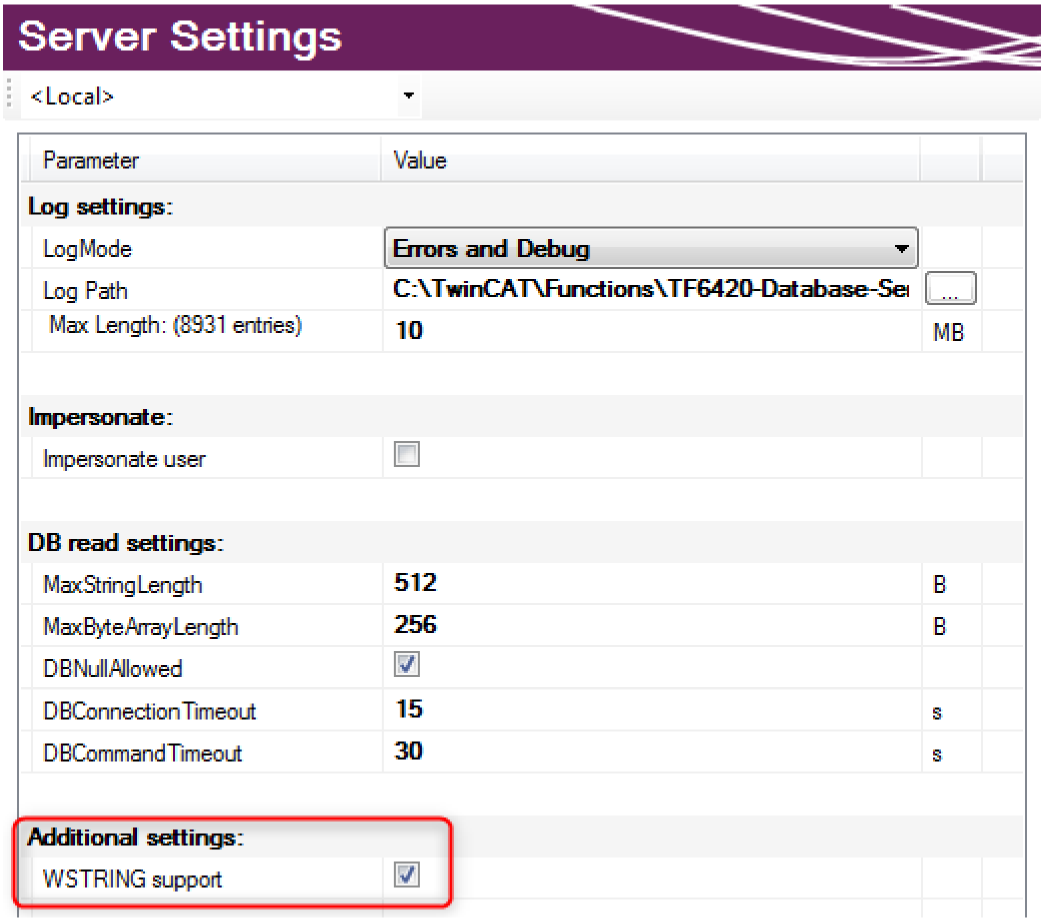
5.2.1 General Information

On the following pages you will find some general information about the supported databases. This information is generic, i.e. not limited to a specific database, and covers topics such as network access, data type support and operating system support.

5.2.1.1 WString support

WSTRING is available for using the Unicode character set.

This data type must be activated in the server settings so that it can be written to the database using the TwinCAT Database Server in PLC Expert mode.

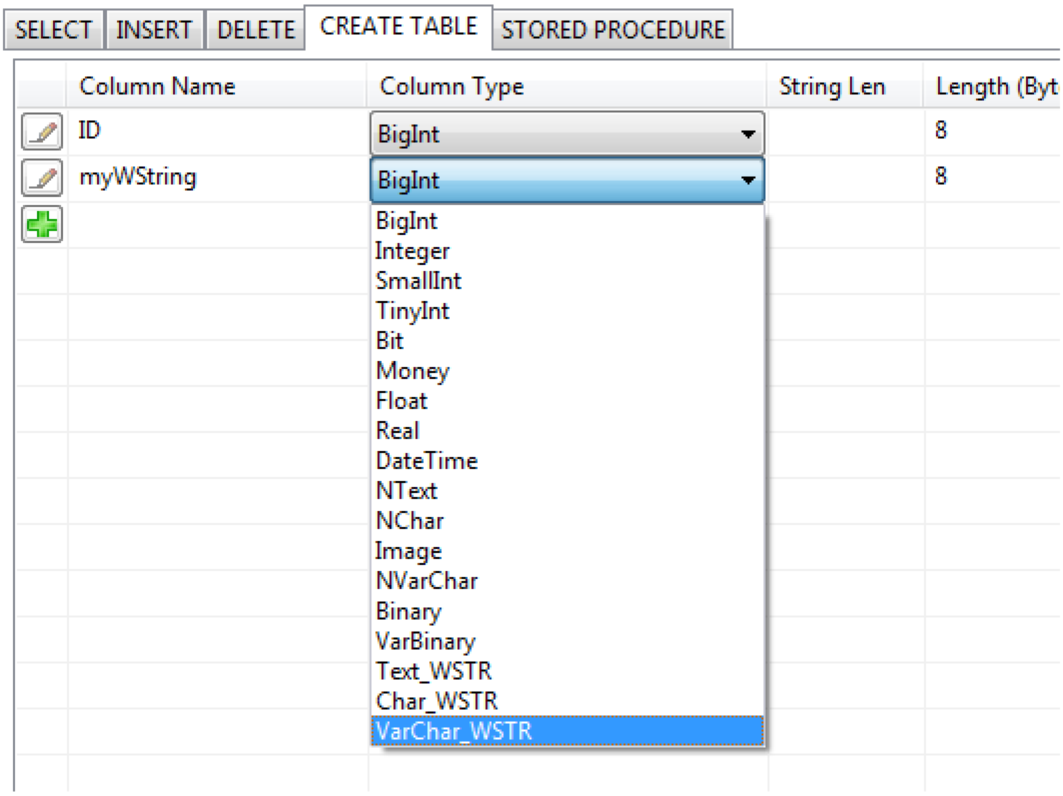


Server Settings

<Local>

Parameter	Value	
Log settings:		
LogMode	Errors and Debug	
Log Path	C:\TwinCAT\Functions\TF6420-Database-Se	...
Max Length: (8931 entries)	10	MB
Impersonate:		
Impersonate user	□	
DB read settings:		
MaxStringLength	512	B
MaxByteArrayLength	256	B
DBNullAllowed	☑	
DBConnectionTimeout	15	s
DBCommandTimeout	30	s
Additional settings:		
WSTRING support	☑	

This data type requires two bytes per character. Please keep this in mind when creating the table structure. To be able to save it in the database in UTF16 format, the column must be created according to the character set. The SQL Query Editor can also be used for this purpose.



Column Name	Column Type	String Len	Length (Byt
ID	BigInt		8
myWString	BigInt		8

The following databases are supported with the Database Server:

Database	UTF8	UTF16	Character set definition	Performance impairment
MySQL	x	x	column-specific	x
MSSQL	x	x	column-specific	
Oracle	x	x	column-specific	
PostgreSQL	x	x	cross-database	x
Others	x			

Performance impairment

Some databases cause performance problems, because additional SQL commands have to be sent in order to read the character set.

WString support is available from version 3.1.31.4.

5.2.1.2 BULK support

The TwinCAT Database Server also supports so-called BULK commands for a selection of databases. BULK commands are SQL statements that insert collected data into multiple rows of a table. Using BULK insert commands usually results in better performance than processes that send a single insert statement to the database for each row to be added.

Currently, BULK commands via the FB_PLCDBCmdEvt function block are supported by the TwinCAT Database Server for Microsoft SQL databases.

Example command: 'SQLBULK<INSERT>#MyTable‘

5.2.2 MS SQL database

This section contains information on the configuration and the data type mapping of Microsoft SQL databases.

Compatible versions: Microsoft SQL database 20xx.

Declaration in the TwinCAT Database Server Configurator

Microsoft SQL database	
Database Type	Select "Microsoft SQL Server" from the drop-down menu.
Provider	"SQLOLEDB"or the provider of the SQL Native Client, e.g. "SQLNCLI10"
Server	Enter the name of your SQL server here. Example: "TESTSERVER \SQLEXPRESS"
Database	Enter the name of the database. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist.
Authentication	Option for logging into the database as a particular user.
User name	Enter the user name here.
Password	Enter the corresponding password.

Windows CE support

This database is also supported by the Windows CE version of the TwinCAT Database Server. The interfacing is not local, but can be established via a network connection.

Data type mapping between DB and PLC

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

Data type support

This database supports the data type WSTRING. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.2.1 Notes on the Microsoft SQL Server

Logs in the Windows Eventlog

Error Event "Report Server Windows Service (SQLEXPRESS) cannot be connected to the report server database."	In the SQL Configuration Manager under SQL Server 2005 stop the SQL Server Reporting Services (SQLEXPRESS) and set the **Start Mode** to "manual". The Database Server doesn't need the Reporting Service.
Information event "'TcDataLogger' database is started"	Open the **Properties** via the context menu in SQL Server Management Studio Express under **Databases/TcDataLogger** and under **Options** set the option **Close automatically** to FALSE. This option is not required because the Database Server opens and closes the database automatically.

It is possible to suppress logging to the Windows Eventlog. Events are then no longer logged. No distinction can be made between the different types of event.

Select the SQL Server (SQLEXPRESS) in the SQL configuration manager under SQL Server 2005 Services and open the **Properties** via the context menu. The **Advanced** tab contains a **Startup parameters** subitem. The individual parameters are separated by semicolons. Add the parameter "-n" and restart the service.

From this point onwards no further events will be logged by the SQL Server.

5.2.3 MS SQL Compact database

This section contains information on the configuration and the data type mapping of Microsoft SQL Compact databases. MS SQL Compact is an ideal database for embedded applications. It has a small footprint but nevertheless provides the required functionality for relational databases.

Compatible versions: Microsoft SQL Compact database 3.5

Declaration in the TwinCAT Database Server Configurator

Microsoft SQL Compact database	
Database Type	Select "Microsoft Compact SQL" from the drop-down menu.
Database URL	Enter the name and path of the database. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist.
Authentication	Option for logging into the database with a password.
Password	Enter the password.

Windows CE support

This database is also supported by the Windows CE version of the TwinCAT Database Server. The connection can be established locally.

Data type mapping between DB and PLC

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

Data type support

WSTRING is not supported by this database. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.4 NET MySQL database

This section contains information on the configuration and the data type mapping of MySQL databases. The NET MySQL driver is an ODBC driver.

Declaration in the TwinCAT Database Server Configurator

MySQL database	
Database Type	Select "NET_MySQL" from the drop-down menu.
Server	Enter the name or IP address of your server here.
Database	Enter the name of the database. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist.
Port	Enter the port for communicating with the MySQL database here. Default: 3306.
User name	Enter the user name here.
Password	Enter the corresponding password.

Windows CE support

This database is also supported by the Windows CE version of the TwinCAT Database Server. The interfacing is not local, but can be established via a network connection.

Data type mapping between DB and PLC

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	DECIMAL(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

Data type support

This database supports the data type WSTRING. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.5 Oracle database

This section contains information on the configuration and the data type mapping of Oracle databases. For interfacing with an Oracle database the so-called ODP driver is used.

Compatible versions: Oracle 9i, 10g, 11g and higher

Note: The TwinCAT Database Server requires the 32-bit version of the .NET ODP components.

Declaration in the TwinCAT Database Server Configurator

Oracle database	
Database Type	Select "Oracle ODP" from the drop-down menu.
Host	Enter the IP or host name of the database.
Service name	Enter the name of the service or the database.
Port	Enter the communication port (optional). Default: 1521.
Protocol	Enter the protocol (optional). Default: TCPIP.
Scheme	Enter the database schema.
User name	Enter the user name here.
Password	Enter the corresponding password.

Windows CE support

Under Windows CE this database is not supported by the TwinCAT Database Server.

Data type mapping between DB and PLC

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

Data type support

This database supports the data type WSTRING. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.6 SQLite

This section contains information on the configuration and the data type mapping of SQLite databases. SQLite is an ideal database for embedded applications. This file-based SQL database requires no installation, since it is already integrated in the TwinCAT Database Server. The relational database offers most of the features of SQL databases and supports the commands of the SQL92 standard. The database enables reliable and fast data storage. However, the database does not allow distinction of users. It is therefore particularly suitable for safe storage of variables on the local system.

Declaration in the TwinCAT Database Server Configurator

SQLite database	
Database Type	Select "SQLite" from the drop-down menu.
SQLite database file	Enter the name and path of the database. You can also use the browser dialog. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist.
Authentication	An option for logging into the database as a particular user.
Password	Enter the corresponding password.

Windows CE support

The database is also supported by the Windows CE version of the TwinCAT Database Server, but only on devices with ARM processor. The connection can be established locally.

Data type mapping between DB and PLC

E_ColumnTypes	SQLite	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

Data type support

This database supports the data type WSTRING. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.7 ASCII-File

Information on configuring ASCII files as databases. The file is generated automatically by the TwinCAT Database Server. A Create Database procedure is not required. The created file can be imported and processed in other spreadsheet programs such as Microsoft Excel.

Declaration in the TwinCAT Database Server Configurator

ASCII file as database	
Database Type	Select "ASCII" from the drop-down menu.
ASCII File	Enter the path for the ASCII file. The file is generated automatically by the TwinCAT Database Server.
Value separator	Here you can specify the separator for the values, i.e. for the columns. Default: ";"
Old ASCII DB format	For compatibility reasons you can optionally switch to the old ASCII format used by the TwinCAT Database Server 3.0.x versions. Use the default table structure.
DBValue Type	Only active if **Old ASCII DB format** is enabled. You can select BYTES or DOUBLE. With DOUBLE the values are in plain text, with BYTES they form a byte stream.

Functions that are not supported

Automatic ID generation is not supported by this database. If the standard table structure is used in Configure mode, the value of the ID is not set.

Data type support

WSTRING is not supported by this database. (See WString support [▶ 119])

Windows CE support

This database is also supported by the Windows CE version of the TwinCAT Database Server. The connection can be established locally.

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.8 XML database

This section contains information on the configuration and the data type mapping of XML files as databases. The database structure, tables and columns are defined in an XSD file. The XML file, the XSD file and an XSL file containing style information are created with the TwinCAT Database Server configurator (**Create** command). Based on the XSL file the XML file can be opened in a web browser, where a graphical enhance view of the database or the table is presented.

Further information on working with XML files as databases can be found in section "XML - information [▶ 128]".

Declaration in the TwinCAT Database Server Configurator

XML database	
Database Type	Select "XML" from the drop-down menu.
XML Database File	Enter the name and path of the XML file.
XML Schema File	Enter the name and path of the XSD file.
Database	Enter the name of the database. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist. In the case of XML databases, the XML, XSD and XSL files are created automatically.

Windows CE support

This database is also supported by the Windows CE version of the TwinCAT Database Server. The connection can be established locally.

Data type mapping between DB and PLC

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

Data type support

WSTRING is not supported by this database. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.8.1 XML - information

1. Using an XML file as a database with the TwinCAT 3 Database Server

2. Apply XPath queries to an XML file with the TwinCAT 3 Database Server

Further information about XML schemas can be found here: http://www.edition-w3.de/TR/2001/REC-xmlschema-0-20010502/

1. XML as database

XSD schema for standard table structure:

```
<?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 file for standard table structure (example):

```
<?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>
```

Data types for XML tables:

```
<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>
```

Data type mapping between DB and PLC

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

Creating/reading of records in/from the XML file

Standard SQL commands can be used for generating records. The TwinCAT Database Server interprets SQL INSERT commands and converts them for the XML file in the form of XML nodes. The TwinCAT Database Server converts SQL SELECT commands for the XML file in the form of XPath queries.

Samples for supported INSERT commands:

- 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);

Samples for supported SELECT commands:

- 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;

Supported function blocks:

- 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 function

XPath types

There are 3 different ways for reading XPath values from an XML file:

- -XPath<ATTR>
  - All attribute values of the selected XML tag are returned to the PLC.
  - If an XML Schema exists, the attributes are converted to the correct data types.
  - If no XML Schema exists, the attributes are returned as T_MaxString.
- -XPath<TAG>
  - The InnerText of the selected XML tag is returned to the PLC.
  - If an XML Schema exists, the value is converted to the correct data type.
  - If no XML Schema exists, the value is returned as T_MaxString.
- -XPath<SUBTAG>
  - The InnerText values of all SubTags of the selected XML tag are returned to the PLC.
  - If an XML Schema exists, the values are converted to the correct data type.
  - If no XML Schema exists, all values are returned as T_MaxString.

Samples:

XML file:

```
<?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>
```

Sample for XPATH<ATTR>

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

Returned structure if **no** schema exists:

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

Returned structure if **one** schema exists:

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

Sample for XPATH<TAG>

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

Returned value if **no** schema exists: SubNode2 : T_MaxString := ‘200’;

Returned value if **one** schema exists: SubNode2 : INT := 200;

Sample for XPATH<SUBTAG>

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

Returned structure if **no** schema exists:

```
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
```

Returned structure if **one** schema exists:

```
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
```

Supported function blocks

- FB_DBRecordSelect
- FB_DBRecordSelect_EX
- FB_DBRecordArraySelect

5.2.9 ODBC databases

Many databases offer ODBC interfaces. The TwinCAT Database Server also has this interface. In the TwinCAT Database Configurator it is therefore generally possible to select an ODBC database in the database configuration menu. So-called "Free Connection Strings [▶ 133]" can be used to form your own connection strings with **Add additional parameter**.

Further known and regularly used ODBC databases are available as templates. These include:

- MySQL [▶ 134]
- Oracle [▶ 136]
- PostgreSQL [▶ 137]
- IBM DB2 [▶ 138]
- Firebird [▶ 139]

Windows CE support

Under Windows CE these databases are not supported by the TwinCAT Database Server.

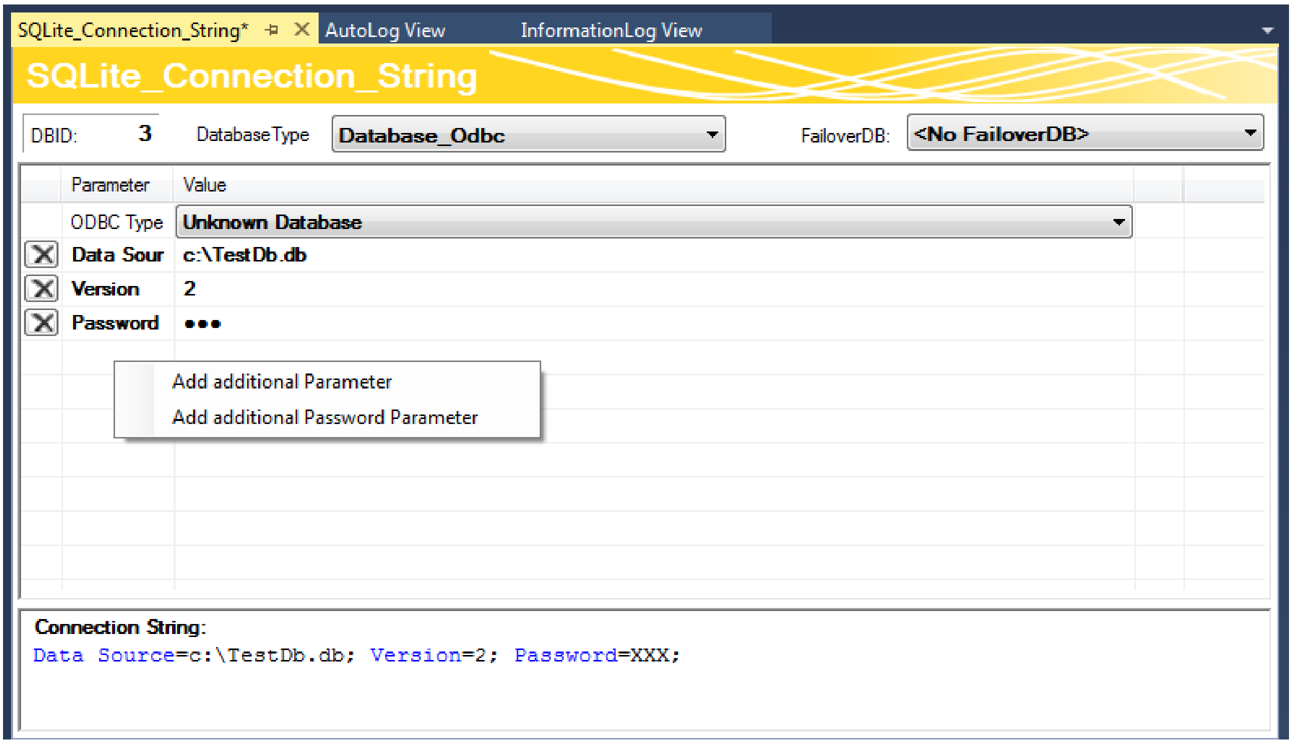
5.2.9.1 Free connection string

To use a database with ODBC interface that is not supported by the TwinCAT Database Server by default, you can select "Unknown Database" as **ODBC type**.

Declaration in the TwinCAT Database Server Configurator

ODBC Free Connection String database	
Database Type	"Odbc_Database"
ODBC Type	Select "Unknown Database" from the drop-down menu.

Simply find the connection string for the ODBC database and remodel it in the configuration window of the TwinCAT Database Server.



DBID: 3 DatabaseType Database_Odbc FailoverDB: <No FailoverDB>

	Parameter	Value
	ODBC Type	Unknown Database
☒	Data Sour	c:\TestDb.db
☒	Version	2
☒	Password	●●●

Connection String:

```
Data Source=c:\TestDb.db; Version=2; Password=XXX;
```

Two commands are available in the context menu for this purpose:

- **Add additional Parameter**
  Adds a general parameter for the connection string. This can have any name, as required by the connection string.
- **Add additional Password Parameter**
  Adds a special password parameter, the value of which is not readable (encrypted) in the configurator and in the configuration file.

Operating principle with Free Connection String

To ensure that the TwinCAT Database Server can work with a Free Connection String, the corresponding driver must be installed on the target system of the TwinCAT Database Server. Only "SQL Expert mode [▶ 19]" can be used!

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.9.2 MySQL

This section contains information on the configuration and the data type mapping of MySQL databases with ODBC.

Declaration in the TwinCAT Database Server Configurator

ODBC MySQL database	
Database Type	“Odbc_Database”
ODBC Type	Select "MySQL" from the drop-down menu.
Driver	Enter the actually installed driver.
Server	Enter the name or IP address of your server.
Database	Enter the name of the database.
Port	Enter the port for communicating with the MySQL database here. Default: 3306.
Option	Default: 2 “Return matched rows instead of affected rows”
Uid	Enter the user name.
Pwd	Enter the corresponding password.

Data type mapping between DB and PLC

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	DECIMAL(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

Data type support

This database supports the data type WSTRING. (See WString support [▸ 119])

Functioning

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [▸ 133]".

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.9.3 Oracle

This section contains information on the configuration and the data type mapping of Oracle databases with ODBC.

Declaration in the TwinCAT Database Server Configurator

ODBC Oracle database	
Database Type	"Odbc_Database"
ODBC Type	Select "Oracle" from the drop-down menu.
Driver	Select here the actually installed driver.
Server	Enter the name or IP address of your server.
Uid	Enter the user name here.
Pwd	Enter the corresponding password.

Data type mapping between DB and PLC

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

Data type support

This database supports the data type WSTRING. (See WString support [▶ 119])

Functioning

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [▶ 133]".

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.9.4 PostgreSQL

This section contains information on the configuration and the data type mapping of PostgreSQL databases with ODBC.

Declaration in the TwinCAT Database Server Configurator

ODBC PostgreSQL database	
Database Type	"Odbc_Database"
ODBC Type	Select "PostgreSQL" from the drop-down menu.
Driver	Select here the actually installed driver.
Server	Enter the name or IP address of your server.
Database	Enter the name of the database.
Port	Enter the port for communicating with the PostgreSQL database. Default: 5432.
Uid	Enter the user name here.
Pwd	Enter here the corresponding password.

Data type mapping between DB and PLC

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

Data type support

This database supports the data type WSTRING. The character set must be set up when the database is created.

Functioning

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [▸ 133]".

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.9.5 IBM DB2

This section contains information on the configuration and the data type mapping of IBM DB2 databases with ODBC.

Declaration in the TwinCAT Database Server Configurator

ODBC IBM DB2 database	
Database Type	"Odbc_Database"
ODBC Type	Select "IBM DB2" from the drop-down menu.
Driver	Enter here the actually installed driver.
Host name	Enter the name or IP address of your server.
Database	Enter the name of the database.
Port	Enter the port for communicating with the IBM DB2 database. Default: 50000.
Protocol	Default: TCPIP
Uid	Enter the user name here.
Pwd	Enter here the corresponding password.
LONGDATACOMPAT	Default: 1

Data type mapping between DB and PLC

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	DECIMAL(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

Data type support

WSTRING is not supported by this database. (See WString support [▸ 119])

Functioning

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [▸ 133]".

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.9.6 Firebird

This section contains information on the configuration and the data type mapping of Firebird databases with ODBC.

Declaration in the TwinCAT Database Server Configurator

ODBC Firebird database	
Database Type	"Odbc_Database"
ODBC Type	Select "Firebird" from the drop-down menu.
Driver	Select here the actually installed driver.
Database	Enter the name of the database.
Client	
Uid	Enter the user name here.
Pwd	Enter the corresponding password.

Data type mapping between DB and PLC

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	DECIMAL(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

Data type support

WSTRING is not supported by this database. (See WString support [▸ 119])

Functioning

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [▸ 133]".

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.10 MS Access database

The values of the variables are saved in a Microsoft Access database.

Access 2000 and Access 2003 (*.mdb) database files are compatible, as are Access 2007 (*.accdb) files. All you have to do is specify different providers in the declaration in the XML configuration file.

Declaration in the TwinCAT Database Server Configurator

Microsoft Access database	
DBValueType	Select "Double" to limit logging to alphanumeric and Boolean data types. Select "Bytes" to also log structures and strings.
DBType	Select "MS Access". PLC: eDBType_Access.
DBServer	Not required.
DBProvider	Access 2000 - Access 2003: The provider is "Microsoft.jet.OLEDB.4.0". Access 2007: The provider is "Microsoft.ACE.OLEDB.12.0".
DBUrl	DBUrl contains the path to the MDB file. e.g. *C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.mdb*
DBTable	DBTable contains the name of the table.

Windows CE support

Under Windows CE this database is not supported by the TwinCAT Database Server.

Data type mapping between DB and PLC

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

Data type support

WSTRING is not supported by this database. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.11 MS Excel database

The variable values are stored in an Microsoft Excel database.

Declaration in the TwinCAT Database Server Configurator

Microsoft Excel database	
DBValueType	Select "Double" to limit logging to alphanumeric and Boolean data types. Select "Bytes" to also log structures and strings.
DBType	Select "MS Excel". PLC: eDBType_MSExcel.
DBServer	Not required.
DBProvider	"Microsoft.Jet.OLEDB.4.0" or "Microsoft.ACE.OLEDB.12.0"
DBUrl	DBUrl contains the path to the Excel file. e.g. *C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.xls*
DBTable	DBTable contains the name of the table.

Windows CE support

Under Windows CE this database is not supported by the TwinCAT Database Server.

Data type mapping between DB and PLC

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)

Functions that are not supported

Automatic ID generation is not supported by this database. If the standard table structure is used in Configure mode, the value of the ID is not set.

Non-supported data types

Binary, VarBinary and Image are not supported with Excel databases.

Data type support

WSTRING is not supported by this database. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

5.2.12 MongoDB

This section contains information on the configuration and the data type mapping of MongoDB databases.

Declaration in the TwinCAT Database Server Configurator

MongoDB	
Database Type	Select "MongoDB" from the drop-down menu.
Server	Enter the name of your MongoDB server.
Database	Enter the name of the database. If the database does not yet exist, it is created the first time it is accessed.
Authentication	**None:** No authentication **User name/password:** Login with user name and password **x509 certificate:** User name: ID of the certificate user Certificate Authority: path to signing certificate (*.crt) Client Certificate: path to client certificate (*.pfx) Client Private Key: password for the client certificate **GSSAPI/Kerberos:** Login with user name and password **LDAP(PLAIN):** Login with user name and password (Since the user name and password are transmitted in plain text, this option is not recommended)

Data type mapping between DB and PLC

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

Data type support

WSTRING is not supported by this database. (See WString support [▶ 119])

NOTE

Data security

In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

MongoDB in PLC Expert mode

PLC Expert mode uses the predefined schema of a database in its function blocks. Normally, the schema of the structures used will not change during operation. In order to nevertheless be able to use the function blocks, the TwinCAT 3 Database Server requires a description of the table schema. A table is simulated. To do this, use the SQL Query Editor to create a table or, in this case, a collection. In addition, unlike for relational databases, an entry is created in a metadata collection. Information on the table schema for the TwinCAT 3 Database Server is stored here.

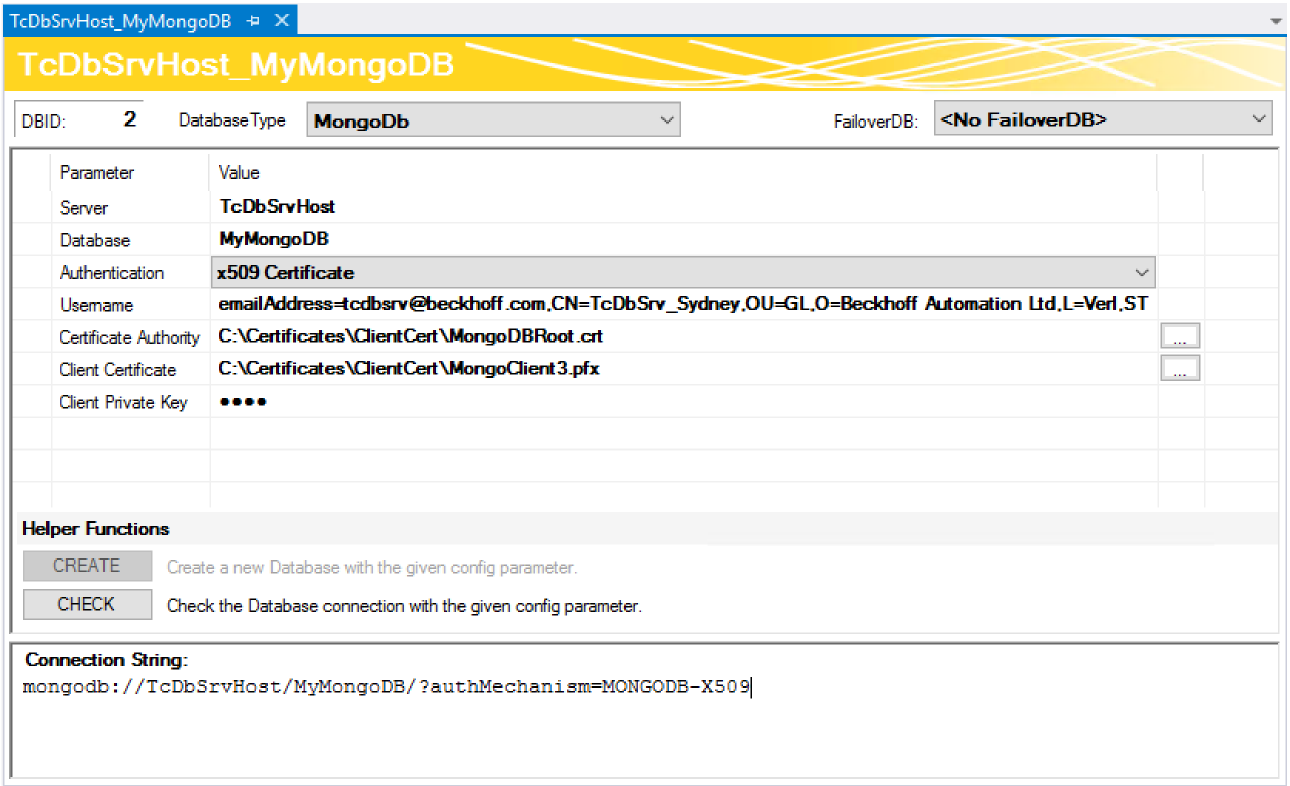
In order to use advanced functionality, e.g. structures of any hierarchy or flexible records, we recommend using the NoSQL function blocks.

Use of certificates

Among other things, MongoDB supports authentication by means of certificates. To this end, select the 'x509 certificate' method under Authentication. The following fields appear:

User name	User name of the corresponding certificate
Certificate Authority	Path to the SSL certificate of the certificate authority. This may be a self-signed certificate.
Client Certificate	Client certificate signed by the SSL certificate.
Client Certificate Password	Password of the client certificate.

Configuring the database connection to MongoDB using certificates:



TcDbSrvHost_MyMongoDB

DBID: 2 Database Type MongoDb FailoverDB: <No FailoverDB>

Parameter	Value
Server	TcDbSrvHost
Database	MyMongoDB
Authentication	x509 Certificate
Username	emailAddress=tcdbsrv@beckhoff.com,CN=TcDbSrv_Sydney,OU=GL,O=Beckhoff Automation Ltd,L=Verl,ST
Certificate Authority	C:\Certificates\ClientCert\MongoDBRoot.crt
Client Certificate	C:\Certificates\ClientCert\MongoClient3.pfx
Client Private Key	●●●●

Helper Functions

CREATE Create a new Database with the given config parameter.

CHECK Check the Database connection with the given config parameter.

Connection String:

```
mongodb://TcDbSrvHost/MyMongoDB/?authMechanism=MONGODB-X509
```

6 PLC API

6.1 Tc3_Database

6.1.1 Function blocks

The function blocks of the Tc3_Database.compiled library are split into three sections, based on the basic concept [▸ 19]:

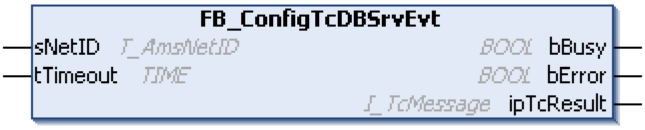
- Configure mode:
  Contains function blocks for controlling reading and writing of AutoLog groups defined in the configurator.
- PLC Expert mode:
  Contains function blocks for conventional PLC programmers.
- SQL Expert mode:
  IT and PLC experts with advanced database knowledge can use these function blocks to assemble SQL commands in the PLC.
- NoSQL Expert mode:
  These function blocks can be used by IT and PLC experts with extended database knowledge to create commands via NoSQL databases and send them to the database.

Using the Tc3_Eventlogger

The TwinCAT 3 Database Server supports the Tc3_Eventlogger API. Further information can be found here [▸ 206] or in the documentation section of the Tc3_Eventlogger.

6.1.1.1 Configure mode

6.1.1.1.1 FB_ConfigTcDBSrvEvt



Function block for creating, reading and deleting configuration entries for the 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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Create [▶ 146]	Local	Creates new entries in the XML configuration file for the TwinCAT Database Server
Read [▶ 147]	Local	Reads the current configuration of the TwinCAT Database Server
Delete [▶ 148]	Local	Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Create

This method creates new entries in the XML configuration file for the TwinCAT Database Server. Optionally the TwinCAT Database Server can use a new entry on a temporary basis. In this case no data is written to the XML file.

Syntax

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

Inputs

Name	Type	Description
pTcDBSrvConfig	POINTER TO BYTE	Pointer of the configuration structure to be created.
cbTcDBSrvConfig	UDINT	Length of the configuration structure
bTemporary	BOOL	Indicates whether the configuration is to be stored in the XML file.
pConfigID	POINTER TO UDINT	Return pointer of the configuration ID (hDBID or hAutoLogGrpID)

Creating AutoLog groups is currently not supported.

Return value

Name	Type	Description
Create	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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 fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Read

This method can be used to read the current configurations of the TwinCAT Database Server. Any temporary configurations that may be included are marked accordingly.

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
```

Inputs

Name	Type	Description
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▸ 233]] OF ST_ConfigDB [▸ 216]	Pointer address of the array into which the database configurations are to be written.
cbDBConfig	UDINT	Length of the database configuration array
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS [▸ 233]] OF ST_ConfigAutoLogGrp [▸ 215]	Pointer address of the array into which the AutoLogGrp configurations are to be written.
cbAutoLogGrpConfig	UDINT	Length of the AutoLogGrp configuration array
pDBCount	POINTER TO UDINT	Pointer address for storing the number of database configurations.
pAutoLogGrpCount	POINTER TO UDINT	Pointer address for storing the number of AutoLogGrp configurations.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Delete

This method can be used to delete databases and AutoLog groups from the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
eTcDBSrvConfigType	E_TcDBSrvConfigType	Type of the configuration to be deleted (database / AutoLog group)
hConfigID	UDINT	ID of the configuration to be deleted (hDBID or hAutoLogGrpID)

Return value

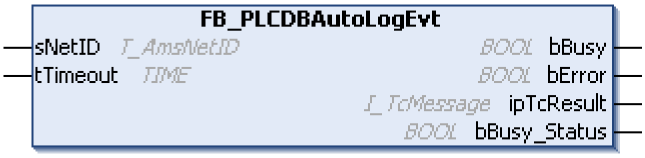
Name	Type	Description
Delete	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block with four methods for starting and stopping of defined AutoLog groups and for reading of the corresponding group status.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active, except for the Status method.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage	Result interface with detailed information on the return value.
bBusy_Status	BOOL	The Status method can be executed independently of the other three methods of the function block and therefore has its own Busy flag. Is TRUE as soon as the Status method is active.

Methods

Name	Definition location	Description
RunOnce [▸ 150]	Local	Executes the AutoLog group once
Start [▸ 151]	Local	Starts AutoLog mode with the corresponding configured AutoLog groups
Status [▸ 151]	Local	Queries the status of the AutoLog groups.
Stop [▸ 152]	Local	Stops AutoLog mode

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

RunOnce

This method can be used to execute an AutoLog group once, for example based on an event in the controller.

Syntax

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

Inputs

Name	Type	Description
hAutoLogGrpID	UDINT	ID of the AutoLog group to be executed once.
bAll	BOOL	If TRUE, all AutoLog groups are executed once.

Return value

Name	Type	Description
RunOnce	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Start

This method starts the AutoLog mode with the corresponding configured AutoLog groups.

Syntax

```
METHOD Start : BOOL
```

Return value

Name	Type	Description
Start	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Status

This method can be used to query the status of the AutoLog groups. A separate busy flag is provided in the body of the function block for this method, since it can be called independently of the other methods of the function block: 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
```

Inputs

Name	Type	Description
tCheckCycle	TIME	Interval time at which the status array is updated.
pError	POINTER TO BOOL	TRUE, if an error has occurred in AutoLog mode.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 233]] OF ST_AutoLogGrpStatus [▶ 230]	Address of the status array that contains all groups.
cbAutoLogStatus	UDINT	Length of the status array

Return value

Name	Type	Description
Status	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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(aAutologG
rpStatus)) THEN
    ; // ...
END_IF
```

Stop

This method stops the AutoLog mode.

Syntax

```
METHOD Stop : BOOL
```

Return value

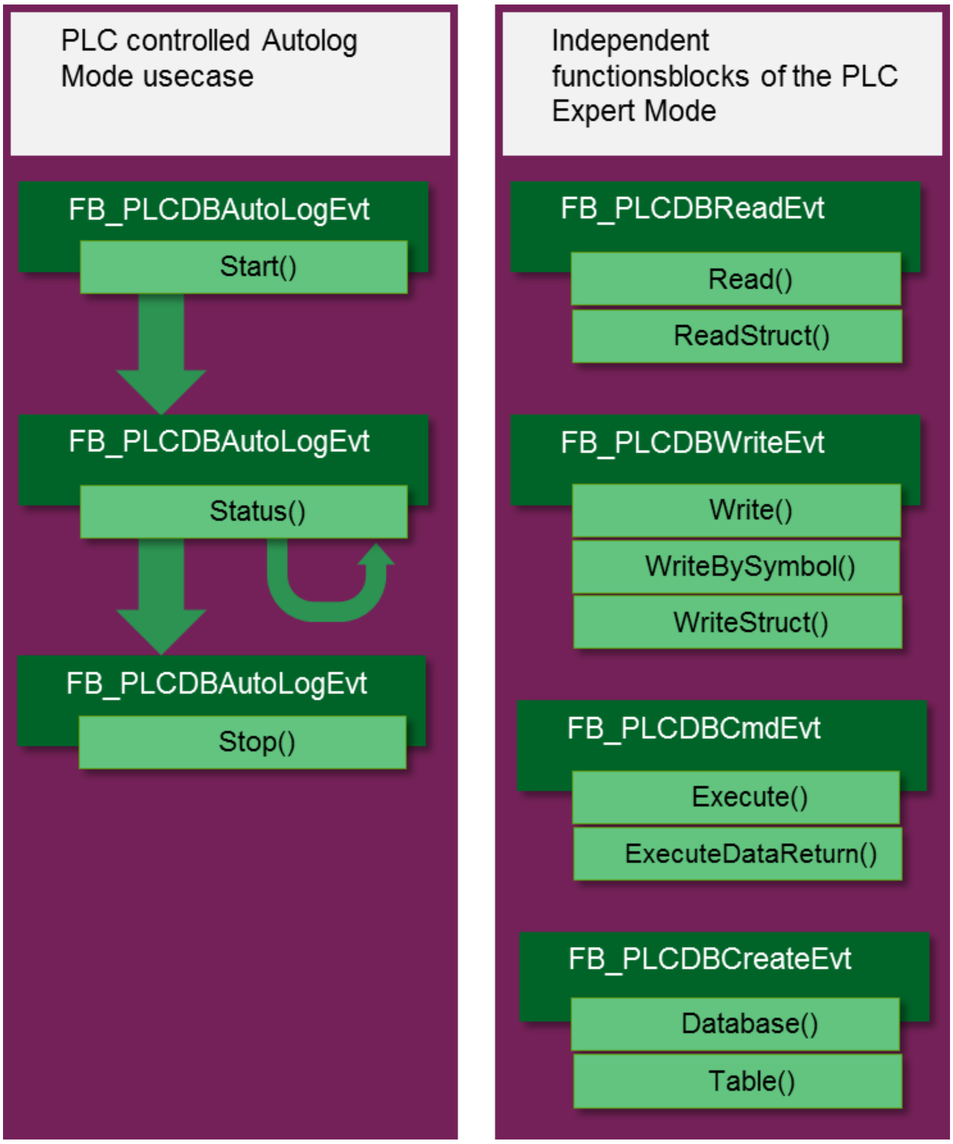
Name	Type	Description
Stop	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

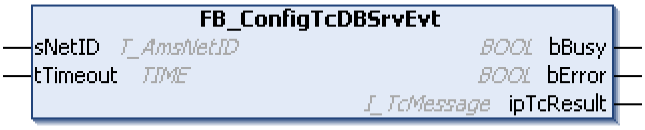
```
VAR
    fbPLCDBAutoLog     : FB_PLCDBAutoLogEvt (sNetID:='', tTimeout := T#5S);
END_VAR
```

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

6.1.1.2 PLC Expert mode



6.1.1.2.1 FB_ConfigTcDBSrvEvt



Function block for creating, reading and deleting configuration entries for the 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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Create [▶ 154]	Local	Creates new entries in the XML configuration file for the TwinCAT Database Server
Read [▶ 155]	Local	Reads the current configuration of the TwinCAT Database Server
Delete [▶ 156]	Local	Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Create

This method creates new entries in the XML configuration file for the TwinCAT Database Server. Optionally the TwinCAT Database Server can use a new entry on a temporary basis. In this case no data is written to the XML file.

Syntax

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

Inputs

Name	Type	Description
pTcDBSrvConfig	POINTER TO BYTE	Pointer of the configuration structure to be created.
cbTcDBSrvConfig	UDINT	Length of the configuration structure
bTemporary	BOOL	Indicates whether the configuration is to be stored in the XML file.
pConfigID	POINTER TO UDINT	Return pointer of the configuration ID (hDBID or hAutoLogGrpID)

Creating AutoLog groups is currently not supported.

Return value

Name	Type	Description
Create	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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 fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Read

This method can be used to read the current configurations of the TwinCAT Database Server. Any temporary configurations that may be included are marked accordingly.

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
```

Inputs

Name	Type	Description
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 233]] OF ST_ConfigDB [▶ 216]	Pointer address of the array into which the database configurations are to be written.
cbDBConfig	UDINT	Length of the database configuration array
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS [▶ 233]] OF ST_ConfigAutoLogGrp [▶ 215]	Pointer address of the array into which the AutoLogGrp configurations are to be written.
cbAutoLogGrpConfig	UDINT	Length of the AutoLogGrp configuration array
pDBCount	POINTER TO UDINT	Pointer address for storing the number of database configurations.
pAutoLogGrpCount	POINTER TO UDINT	Pointer address for storing the number of AutoLogGrp configurations.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Delete

This method can be used to delete databases and AutoLog groups from the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
eTcDBSrvConfigType	E_TcDBSrvConfigType	Type of the configuration to be deleted (database / AutoLog group)
hConfigID	UDINT	ID of the configuration to be deleted (hDBID or hAutoLogGrpID)

Return value

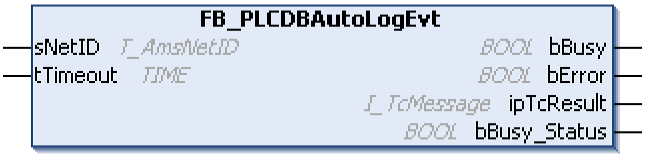
Name	Type	Description
Delete	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block with four methods for starting and stopping of defined AutoLog groups and for reading of the corresponding group status.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active, except for the Status method.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage	Result interface with detailed information on the return value.
bBusy_Status	BOOL	The Status method can be executed independently of the other three methods of the function block and therefore has its own Busy flag. Is TRUE as soon as the Status method is active.

Methods

Name	Definition location	Description
RunOnce [▶ 158]	Local	Executes the AutoLog group once
Start [▶ 159]	Local	Starts AutoLog mode with the corresponding configured AutoLog groups
Status [▶ 159]	Local	Queries the status of the AutoLog groups.
Stop [▶ 160]	Local	Stops AutoLog mode

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

RunOnce

This method can be used to execute an AutoLog group once, for example based on an event in the controller.

Syntax

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

Inputs

Name	Type	Description
hAutoLogGrpID	UDINT	ID of the AutoLog group to be executed once.
bAll	BOOL	If TRUE, all AutoLog groups are executed once.

Return value

Name	Type	Description
RunOnce	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Start

This method starts the AutoLog mode with the corresponding configured AutoLog groups.

Syntax

```
METHOD Start : BOOL
```

Return value

Name	Type	Description
Start	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Status

This method can be used to query the status of the AutoLog groups. A separate busy flag is provided in the body of the function block for this method, since it can be called independently of the other methods of the function block: 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
```

Inputs

Name	Type	Description
tCheckCycle	TIME	Interval time at which the status array is updated.
pError	POINTER TO BOOL	TRUE, if an error has occurred in AutoLog mode.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 233]] OF ST_AutoLogGrpStatus [▶ 230]	Address of the status array that contains all groups.
cbAutoLogStatus	UDINT	Length of the status array

Return value

Name	Type	Description
Status	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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(aAutologG
rpStatus)) THEN
    ; // ...
END_IF
```

Stop

This method stops the AutoLog mode.

Syntax

```
METHOD Stop : BOOL
```

Return value

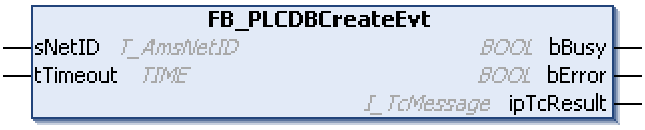
Name	Type	Description
Stop	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

6.1.1.2.3 FB_PLCDBCreateEvt



Function block with two methods. One method can be used to create databases from the PLC on a database server specified in the PLC. The other method can be used to generate a new table in a specified database.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Database [▶ 162]	Local	Creates a new database
Table [▶ 163]	Local	Creates a new table with a structure that is defined via an array with x elements or x columns in the PLC.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Database

This method creates a new database. Optionally you can specify whether the created database should also be used for the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
pDatabaseConfig	POINTER TO BYTE	Address of the database configuration structure [▶ 217]
cbDatabaseConfig	UDINT	Length of the database configuration structure
bCreateXMLConfig	BOOL	Indicates whether the newly created database should be entered as new configuration entry in the XML file.
pDBID	UDINT	Returns the hDBID if/when a new configuration entry was created.

Return value

Name	Type	Description
Database	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Table

This method creates a new table with a structure that is defined through an array with x elements or x columns in the PLC.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	MaxString	Name of the table to be created.
pTableCfg	POINTER TO ARRAY[0..MAX_DBCOLUMNS [▶ 233]] OF ST_ColumnInfo [▶ 231]	Indicates the pointer address of the table structure array. The individual columns are written in this array.
cbTableCfg	UDINT	Indicates the length of the array in which the columns are configured.

Return value

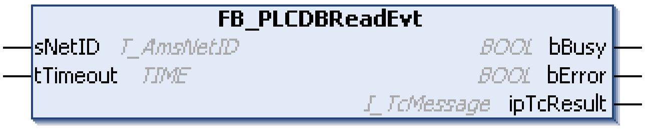
Name	Type	Description
Table	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block for reading records from a database.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▸ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Read [▸ 165]	Local	Reads a specified number of records from a database table with the standard table structure specified by Beckhoff.
ReadStruct [▸ 166]	Local	Reads a specified number of records from a database table with any table structure.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Read

This method reads a specified number of records from a database table with the standard table structure specified by Beckhoff. The standard table structure is used in AutoLog mode and in the FB_DBWriteEvt function block, for example.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
sDBSymbolName	T_MaxString	Symbol name to be read from the standard table structure.
eOrderBy	E_OrderColumn.eColumnID	Sorting column (ID, timestamp, name or value)
eOrderType	E_OrderType.eOrder_ASC	Sorting direction (ASC or DESC)
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO ST_StandardRecord	Address of the structure array into which the records are to be written.
cbData	UDINT	Indicates the size of the structure array in bytes.

Return value

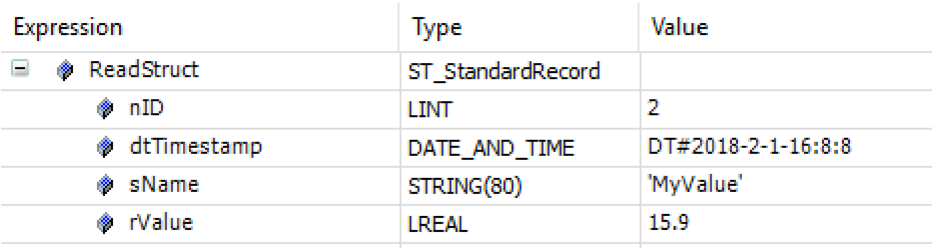
Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Result in the PLC:



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

ReadStruct

This method reads a specified number of records from a database table with any table structure.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Address of the array containing the column name to be read.
cbColumnNames	UDINT	Length of the column name array
sOrderByColumn	STRING(50)	Name the sorting column
eOrderType	E_OrderType	Sorting direction (ASC or DESC)
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO BYTE	Address of the structure array into which the records are to be written.
cbData	UDINT	Indicates the size of the structure array in bytes.

Return value

Name	Type	Description
ReadStruct	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

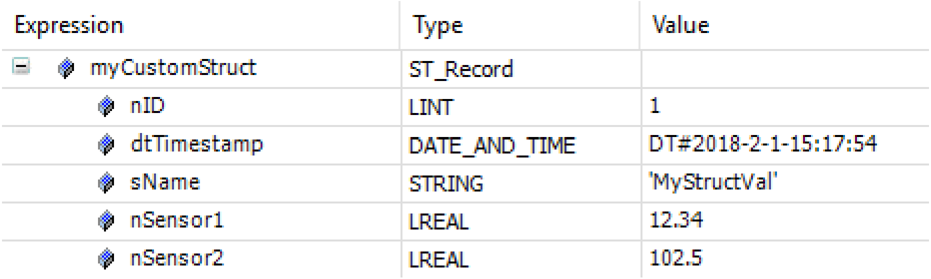
Sample

```
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
```

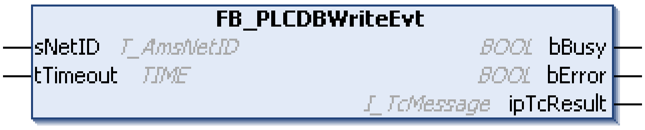
```
// set columnnames
ColumnNames[0]  := 'ID';
ColumnNames[1]  := 'Timestamp';
ColumnNames[2]  := 'Name';
ColumnNames[3]  := 'Sensor1';
ColumnNames[4]  := 'Sensor2';

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
```

Result in the PLC:



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



Function block for writing of records into a database.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Write [▶ 169]	Local	Creates a record in the standard table structure specified by Beckhoff.
WriteBySymbol [▶ 170]	Local	Reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff.
WriteStruct [▶ 171]	Local	Creates a record with any table structure.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Write

This method creates a record in the standard table structure specified by Beckhoff.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
pValue	POINTER TO BYTE	Address of the variable to be logged in the standard table structure.
cbValue	UDINT	Length of the variable to be logged.
sDBSymbolName	T_MaxString	Name that is logged in the table.
eDBWriteMode	E_WriteMode	Indicates the write mode. (append, update, ring buffer)
nRingBuffParameter	UDINT	Additional parameter(s) for the "ring buffer" write mode.

Return value

Name	Type	Description
Write	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

This sample shows how to use the FB_PLCDBWriteEvt.Write method:

```
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
```

Result in the database:

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

With the ring buffer option, only three entries of this name are in the database at any one time. Older entries are deleted.

WriteBySymbol

This method reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff. ADS symbols from other ADS devices can also be read.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
stADSDevice	ST_ADSDevice	ADS device from which a symbol is to be logged in the standard table structure.
stSymbol	ST_Symbol	Symbol name of the variable to be written
eDBWriteMode	E_WriteMode	Indicates the write mode. (append, update, ring buffer)
nRingBuffParameter	UDINT	Additional parameter(s) for the "ring buffer" write mode

Return value

Name	Type	Description
WriteBySymbol	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

This sample shows how to use the FB_PLCDBWriteEvt.WriteBySymbol method:

```
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
```

Result in the database:

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

WriteStruct

This method creates a record with a freely selectable table structure.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
pRecord	POINTER TO BYTE	Address of a structure that is to be logged in a freely selectable table structure.
cbRecord	UDINT	Length of the structure to be written
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Address of the array containing the column name to be filled.
cbColumnNames	UDINT	Length of the column name array

Return value

Name	Type	Description
WriteStruct	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

This sample shows how to use the method 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
```

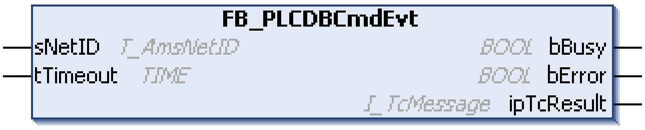
```
// set Values
systime(timeLoDw => currentTime.dwLowDateTime, timeHiDW => currentTime.dwHighDateTime );
myRecord.dtTimestamp := FILETIME_TO_DT(currentTime);
myRecord.sName       := 'MyStructVal';
myRecord.nSensor1    := 12.34;
myRecord.nSensor2    := 102.5;

// 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
```

Result in the database:

ID	Timestamp	Name	Sensor1	Sensor2
5	'2018-01-30 15:23:26'	‘MyStructVal’	12.34	102.5

6.1.1.2.6 FB_PLCDBCmdEvt



Function block with two methods. Users can define and transfer their own SQL commands. Placeholders in the SQL command can correlate with structures in the PLC, which reflect the table structure. The database server enters the current data of the structure into the SQL command.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Execute [▶ 174]	Local	Sends any SQL commands to the database. Returned records cannot be read.
ExecuteDataReturn [▶ 175]	Local	Sends any SQL commands to the database. A specified number of records can be read.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Execute

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are replaced by the TwinCAT Database Server with the corresponding values before the execution. Returned records cannot be read.

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
```

Sample

```
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
```

ExecuteDataReturn

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are replaced by the TwinCAT Database Server with the corresponding values before the execution. A specified number of records can be read.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
pExpression	POINTER TO BYTE	Address of the string variable with the SQL command
cbExpression	UDINT	Length of the string variable with the SQL command
pData	POINTER TO BYTE	Address of the structure with the parameter values
cbData	UDINT	Length of the structure with the parameter values
pParameter	POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter	Address of the structure array with the parameter information
cbParameter	UDINT	Length of the structure array with the parameter information
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pReturnData	POINTER TO BYTE	Address of the structure array into which the records are to be written.
cbReturnData	UDINT	Indicates the size of the structure array in bytes.
pRecords	POINTER TO BYTE	Number of read records.

Return value

Name	Type	Description
ExecuteDataReturn	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Parameterizing the command

The column names for the individual parameters are specified in curly brackets in the SQL command.
Sample: ‚SELECT * FROM MyHouse_Temperatures WHERE Room = {SelectedRoom}'.
Accordingly, SelectedRoom has to be specified as parameter name in the structure ST_ExpParameter.

Some databases do not support the parameterization of SQL clauses. (TOP/LIMIT/ROWNUM/...)
Parameterizable table names are not usually supported.

Sample

```
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

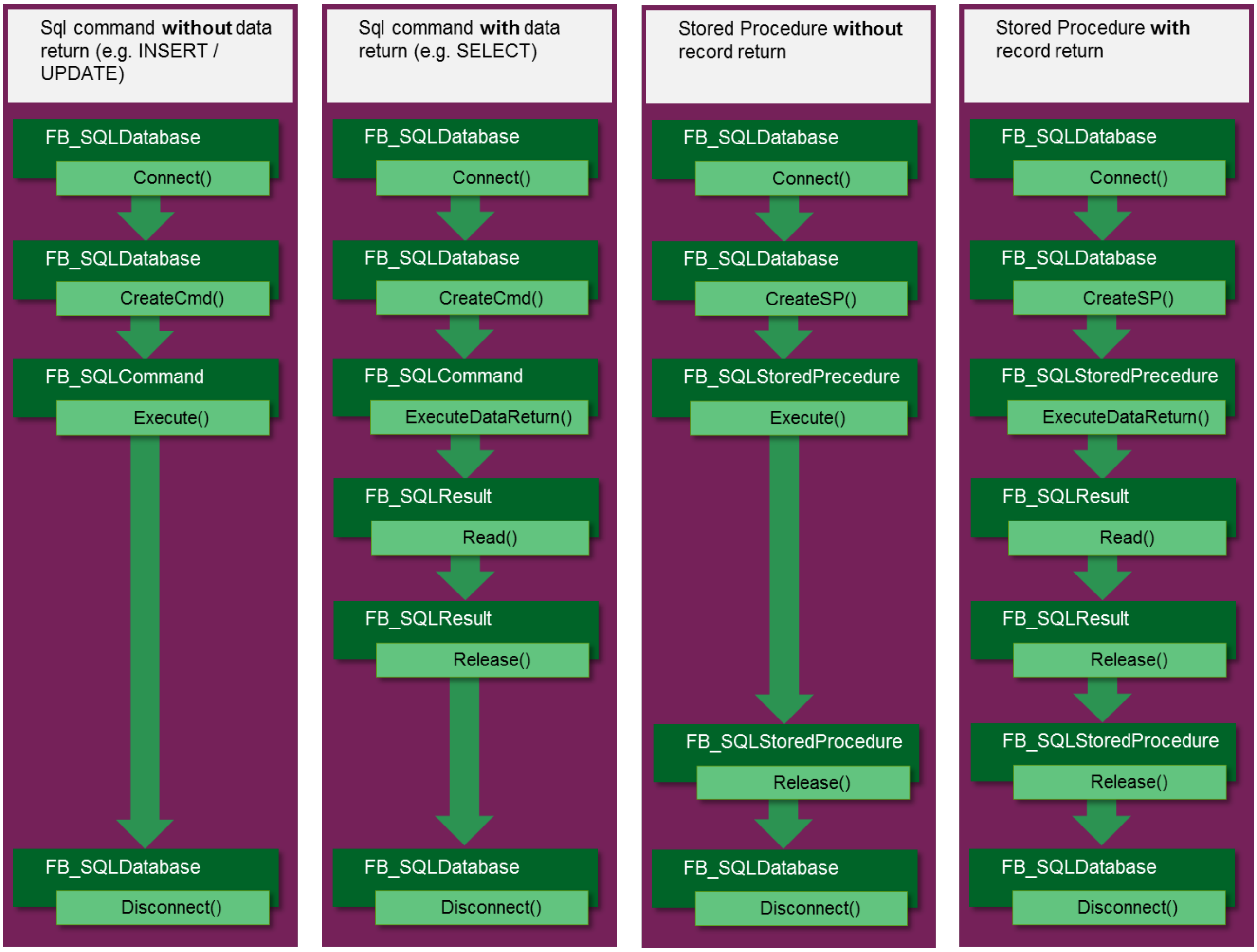
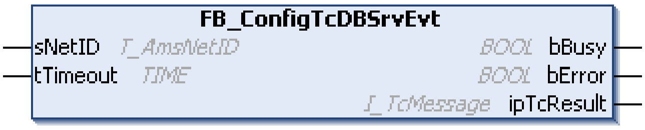


Fig. 1:

6.1.1.3.1 FB_ConfigTcDBSrvEvt



Function block for creating, reading and deleting configuration entries for the 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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Create [▶ 179]	Local	Creates new entries in the XML configuration file for the TwinCAT Database Server
Read [▶ 180]	Local	Reads the current configuration of the TwinCAT Database Server
Delete [▶ 181]	Local	Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Create

This method creates new entries in the XML configuration file for the TwinCAT Database Server. Optionally the TwinCAT Database Server can use a new entry on a temporary basis. In this case no data is written to the XML file.

Syntax

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

Inputs

Name	Type	Description
pTcDBSrvConfig	POINTER TO BYTE	Pointer of the configuration structure to be created.
cbTcDBSrvConfig	UDINT	Length of the configuration structure
bTemporary	BOOL	Indicates whether the configuration is to be stored in the XML file.
pConfigID	POINTER TO UDINT	Return pointer of the configuration ID (hDBID or hAutoLogGrpID)

Creating AutoLog groups is currently not supported.

Return value

Name	Type	Description
Create	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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 fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Read

This method can be used to read the current configurations of the TwinCAT Database Server. Any temporary configurations that may be included are marked accordingly.

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
```

Inputs

Name	Type	Description
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS [▶ 233]] OF ST_ConfigDB [▶ 216]	Pointer address of the array into which the database configurations are to be written.
cbDBConfig	UDINT	Length of the database configuration array
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS [▶ 233]] OF ST_ConfigAutoLogGrp [▶ 215]	Pointer address of the array into which the AutoLogGrp configurations are to be written.
cbAutoLogGrpConfig	UDINT	Length of the AutoLogGrp configuration array
pDBCount	POINTER TO UDINT	Pointer address for storing the number of database configurations.
pAutoLogGrpCount	POINTER TO UDINT	Pointer address for storing the number of AutoLogGrp configurations.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Delete

This method can be used to delete databases and AutoLog groups from the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
eTcDBSrvConfigType	E_TcDBSrvConfigType	Type of the configuration to be deleted (database / AutoLog group)
hConfigID	UDINT	ID of the configuration to be deleted (hDBID or hAutoLogGrpID)

Return value

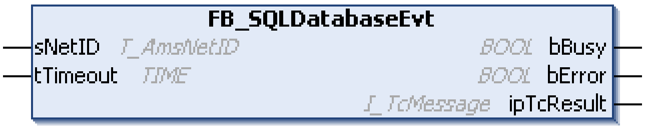
Name	Type	Description
Delete	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block for opening, closing and managing a database connection.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Connect [▶ 183]	Local	Opens a connection to a declared database.
CreateCmd [▶ 183]	Local	Initializes an instance of the function block FB_SQLCommandEvt [▶ 185] with the already open database connection of the function block FB_SQLDatabaseEvt.
CreateSP [▶ 184]	Local	Initializes an instance of the function block FB_SQLStoredProcedureEvt [▶ 191] with the already open database connection of the function block FB_SQLDatabaseEvt.
Disconnect [▶ 185]	Local	Closes the connection to the database that was opened by this function block instance.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Connect

This method opens a connection to a declared database.

Syntax

```
METHOD Connect : BOOL
VAR_INPUT
    hDBID: UDINT := 1;
END_VAR
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.

Return value

Name	Type	Description
Connect	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

CreateCmd

This method is used to initialize an instance of the function block FB_SQLCommand with the already open database connection of the function block FB_SQLDatabase. The function block FB_SQLCommand only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLCommand can be initialized with the same database connection.

The initialization of the function block FB_SQLCommand is completed in the same cycle. This means that neither the Busy flag of the function block nor the method return value of the CreateCmd method have to be checked.

Syntax

```
METHOD CreateCmd : BOOL
VAR_INPUT
    pSQLCommand: POINTER TO FB_SQLCommandEvt;
END_VAR
```

Inputs

Name	Type	Description
pSQLCommand	POINTER TO FB_SQLCommand	Returns a new instance of the function block FB_SQLCommandEvt.

Return value

Name	Type	Description
CreateCmd	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

FB_SQLCommandEvt [▶ 185] can then be used for the execution.

CreateSP

This method is used to initialize an instance of the function block FB_SQLStoredProcedureEvt with the already open database connection of the function block FB_SQLDatabaseEvt. The function block FB_SQLStoredProcedureEvt only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLStoredProcedureEvt can be initialized with the same database connection.

The initialization of the function block FB_SQLStoredProcedureEvt may take several cycles. The Busy flag of the function block or the method return value of the CreateCmd method have to be checked before the function block can be used.

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
```

Inputs

Name	Type	Description
sProcedureName	T_MaxString	Indicates the name of the procedure to be executed.
pParameterInfo	POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter	Pointer address for the parameter info list.
cbParameterInfo	UDINT	Indicates the length of the parameter info list.
pSQLProcedure	POINTER TO FB_SQLStoredProcedureEvt	Returns a new instance of the function block FB_SQLStoredProcedureEvt.

Return value

Name	Type	Description
CreateSP	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Subsequently, the FB_SQLStoredProcedureEvt [▶ 191] can be used to execute the stored procedure.

Disconnect

This method closes the connection to the database that was opened by this function block instance.

Syntax

```
METHOD Disconnect : BOOL
```

Return value

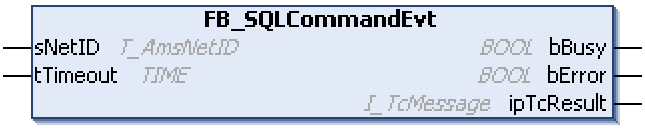
Name	Type	Description
Disconnect	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block for executing SQL commands. Before it can be used it has to be initialized with the function block FB_SQLDatabaseEvt.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Execute [▶ 186]	Local	Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabaseEvt [▶ 181].
ExecuteDataReturn [▶ 187]	Local	Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabaseEvt [▶ 181]. An instance of the function block FB_SQLResultEvt [▶ 188] can be transferred for reading the returned records.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Execute

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
END_VAR
```

Inputs

Name	Type	Description
pSQLCmd	POINTER TO BYTE	Indicates the pointer address of a string variable with the SQL command to be executed.
cbSQLCmd	UDINT	Indicates the length of a SQL command to be executed.

Return value

Name	Type	Description
Execute	POINTER TO BYTE	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

Uses the command created by FB_SQLDatabaseEvt.CreateCmd() [▶ 181].

```
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
```

ExecuteDataReturn

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the function block FB_SQLResult can be transferred for reading the returned records.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
    pSQLDBResult: POINTER TO FB_SQLResult;
END_VAR
```

Inputs

Name	Type	Description
pSQLCmd	POINTER TO BYTE	Indicates the pointer address of a string variable with the SQL command to be executed.
cbSQLCmd	UDINT	Indicates the length of a SQL command to be executed.
pSQLDBResult	POINTER TO FB_SQLResult [▸ 188]	Returns a new instance of the function block FB_SQLResult.

Return value

Name	Type	Description
ExecuteDataReturn	POINTER TO BYTE	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

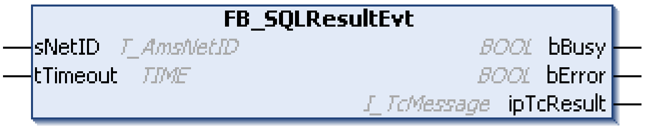
Uses the command created by FB_SQLDatabaseEvt.CreateCmd() [▸ 181].

```
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
```

FB_SQLResultEvt [▸ 188] can then be used to read the data.

6.1.1.3.4 FB_SQLResultEvt



The function block is used for reading the cached records.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Read [▶ 189]	Local	Reads a specified number of records from the result data cached in the TwinCAT Database Server.
Release [▶ 190]	Local	Releases data buffered by the TwinCAT Database Server.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Read

This method reads a specified number of records from the result data cached in the TwinCAT Database Server.

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
```

Inputs

Name	Type	Description
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO BYTE	Address of the structure array into which the records are to be written.
cbData	UDINT	Indicates the size of the structure array in bytes.
bWithVerifying	BOOL	Return data are compared with the pData structure array and adjusted if necessary.
bDataRelease	BOOL	Releases the cached data.

Return value

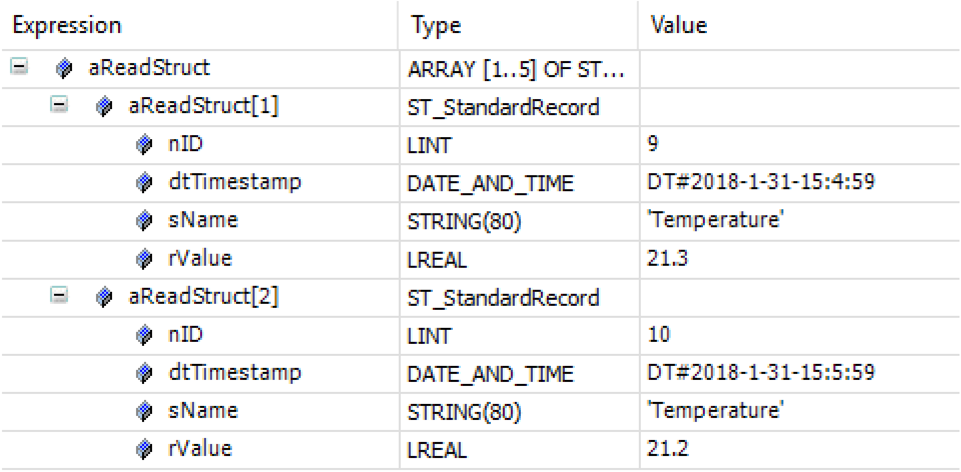
Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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), TRUE, TRUE) THEN
    IF fbSqlResult.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Result in the PLC:



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

Release

This method can be used to release data cached by the TwinCAT Database Server.

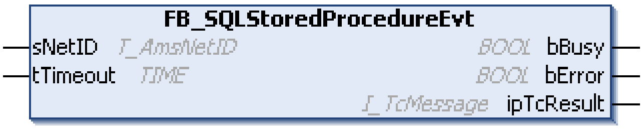
Syntax

```
METHOD Release : BOOL
```

Return value

Name	Type	Description
Release	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

6.1.1.3.5 FB_SQLStoredProcedureEvt



Function block for executing stored procedures of the database. Before it can be used it has to be initialized with the function block FB_SQLDatabaseEvt.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
Execute [▶ 192]	Local	Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabaseEvt [▶ 181].
ExecuteDataReturn [▶ 193]	Local	Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabaseEvt [▶ 181]. An instance of the function block FB_SQLResultEvt [▶ 188] can be transferred for reading the returned records.
Release [▶ 193]	Local	Releases the parameter information of the stored procedure that was transferred during initialization.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Execute

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabaseEvt.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
END_VAR
```

Inputs

Name	Type	Description
pParameterStrc	POINTER TO BYTE	Pointer address to the parameter structure that is transferred to the procedure.
cbParameterStrc	UDINT	Length of the parameter structure.

Return value

Name	Type	Description
Execute	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [▶ 181].

```
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
```

ExecuteDataReturn

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the FB_SQLResult function block can be transferred for reading the returned records.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
    pSQLDBResult: POINTER TO FB_SQLDBResultEvt;
END_VAR
```

Inputs

Name	Type	Description
pParameterStrc	POINTER TO BYTE	Pointer address to the parameter structure that is transferred to the procedure.
cbParameterStrc	UDINT	Length of the parameter structure
pSQLDBResult	POINTER TO FB_SQLDBResultEvt	Returns a new instance of the function block FB_SQLDBResultEvt.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [▶ 181].

```
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
```

FB_SQLResultEvt [▶ 188] can then be used to read the data.

Release

This method releases the parameter information of the stored procedure, which was transferred during initialization.

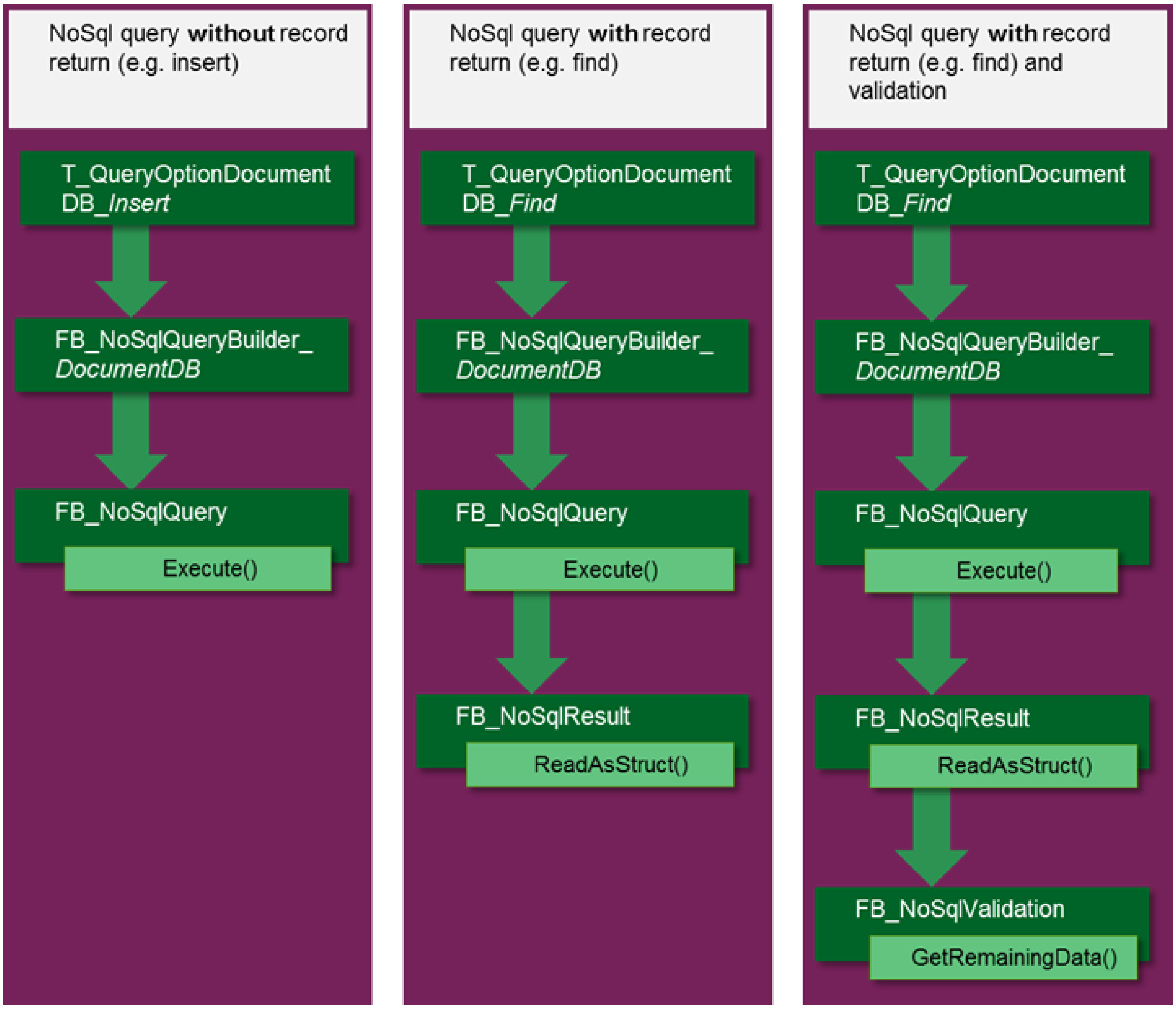
Syntax

```
METHOD Release : BOOL
```

Return value

Name	Type	Description
Release	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

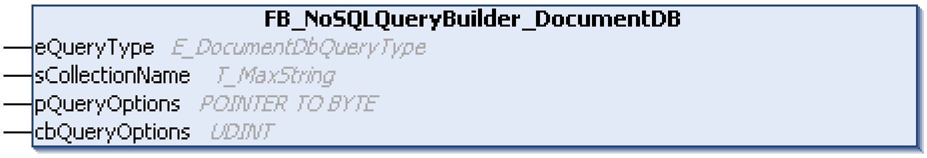
6.1.1.4 NoSQL Expert Mode


NoSql query **without** record return (e.g. insert)
T_QueryOptionDocument DB_*Insert*
FB_NoSqlQueryBuilder_ *DocumentDB*
FB_NoSqlQuery
Execute()
NoSql query **with** record return (e.g. find)
T_QueryOptionDocument DB_*Find*
FB_NoSqlQueryBuilder_ *DocumentDB*
FB_NoSqlQuery
Execute()
FB_NoSqlResult
ReadAsStruct()
NoSql query **with** record return (e.g. find) and validation
T_QueryOptionDocument DB_*Find*
FB_NoSqlQueryBuilder_ *DocumentDB*
FB_NoSqlQuery
Execute()
FB_NoSqlResult
ReadAsStruct()
FB_NoSqlValidation
GetRemainingData()

6.1.1.4.1 Query Builder

In order to support as many NoSQL databases as possible, there are query function blocks that offer different parameterizations for queries. These function blocks are then passed as an interface to the methods of the FB_NoSQLQuery.

FB_NoSQLQueryBuilder_DocumentDB



```
FB_NoSQLQueryBuilder_DocumentDB
eQueryType      E_DocumentDbQueryType
sCollectionName   T_MaxString
pQueryOptions   POINTER TO BYTE
cbQueryOptions   UDINT
```

Function block for defining a query for the database. The query is sent with FB_NoSQLQueryEvt [▶ 196]. It is not necessary to call the Build method.

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
```

Inputs

Name	Type	Description
eQueryType	E_DocumentDbQueryType [▶ 222]	Type of query sent to the database.
sCollectionName	T_ MAXSTRING	Name of the collection that is the target of the query.
pQueryOptions	POINTER TO BYTE	Specifies the address for the query options [▶ 223].
cbQueryOptionsr	UDINT	Length of the query options.

Methods

Name	Definition location	Description
Build [▶ 196]	Local	[optional] This method generates a query for the function block FB_NoSQLQueryEvt [▶ 196] from the set parameters.

Sample:

```
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);
```

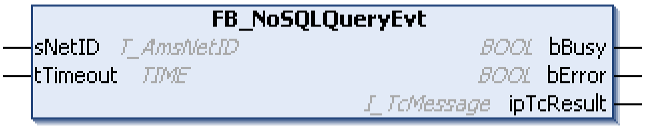
Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Build

This method is called automatically in case of a FB_NoSQLQuery [▸ 196]Evt (either with Execute or ExecuteDataReturn) before the query is sent. It creates a TwinCAT 3 Database Server-specific query from the specified parameters of the QueryBuilder.

6.1.1.4.2 FB_NoSQLQueryEvt



Function block for executing NoSQL database queries. A QueryBuilder function block that describes the query is used as the input parameter for the methods.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▸ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▸ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Methods

Name	Definition location	Description
Execute [▸ 197]	Local	Sends the query created by the QueryBuilder function block to the database.
ExecuteDataReturn [▸ 198]	Local	Sends the query created by the QueryBuilder function block to the database. An instance of the FB_NoSqlResultEvt [▸ 199] function block can be transferred for reading the returned records.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

Execute

This method sends a query to the NoSQL database, which was previously set with the I_NoSQLQueryBuilder [▸ 194] function block.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    hDBID: UDINT;
    iNoSQLQueryBuilder: I_NoSQLQueryBuilder;
END_VAR
```

Inputs

Name	Type	Description
hDBID	UDINT	ID of the set database configuration
iNoSSQLQuery Builder	I_NoSQLQueryBuilder	Pre-parameterized QueryBuilder function block. This varies depending on the database.

Return value

Name	Type	Description
Execute	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Uses the QueryBuilder to execute the corresponding query.

```
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
```

First, the FB_NoSQLQueryEvt function block is parameterized via the FB_NoSQLQueryBuilder_DocumentDB [▸ 195] function block. Depending on the query type there are various options, such as T_QueryOptionDocumentDB_Insert [▸ 224], for setting the document to be inserted.

ExecuteDataReturn

This method executes a query to a NoSQL database that was previously set using the I_NoSQLQueryBuilder function block. The transferred instance of type FB_NoSQLResultEvt is filled with return values.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    hDBID : UDINT;
    iNoSSQLQueryBuilder: I_NoSQLQueryBuilder;
    pNoSQLResult: POINTER TO FB_NoSQLResultEvt;
END_VAR
```

Inputs

Name	Type	Description
hDBID	UDINT	ID of the set database configuration
iNoSQLQueryBuilder	I_NoSQLQueryBuilder	Preconfigured QueryBuilder function block that defines the query to be sent.
pNoSQLResult	POINTER TO FB_NoSSQLResultEvt	Specifies the address for the FB_NoSQLResultEvt function block, which can be used to read the results.

Return value

Name	Type	Description
ExecuteDataReturn	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.
nDataCount	UDINT	[optional] Number of records returned

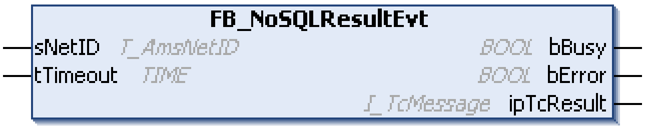
Uses the QueryBuilder to execute the corresponding query.

```
VAR
    fbNoSqlQuery : FB_NoSSQLQueryEvt(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
```

First, the FB_NoSQLQueryEvt function block is parameterized via the FB_NoSQLQueryBuilder_DocumentDB [▶ 195] function block. Depending on the query type there are various options, such as T_QueryOptionDocumentDB_Find [▶ 223], for defining the filter, sorting or projection.

6.1.1.4.3 FB_NoSQLResultEvt



Function block for reading buffered records.

The records must first be retrieved from the database using the function block FB_NoSQLQueryEvt [▶ 196] when the ExecuteDataReturn [▶ 198] method is called. The function block FB_NoSQLResultEvt is specified for initialization. They can then be read out either as a PLC structure or as a string.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.
nDataCount	UDINT	Indicates the number of returned records available from the call of the function block FB_NoSQLQueryEvt.ExecuteDataReturn() [▶ 198].

Methods

Name	Definition location	Description
ReadAsString [▶ 200]	Local	Reads a specified number of records from the result data cached in the TwinCAT Database Server as JSON string.
ReadAsStruct [▶ 201]	Local	Reads a specified number of records from the result data cached in the TwinCAT Database Server into the specified structure.
Release [▶ 202]	Local	Releases data buffered by the TwinCAT Database Server.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

ReadAsString

This method reads a specified number of records from the result data cached in the TwinCAT Database Server. An array of strings is specified into which this data is to be copied as JSON.

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
```

Inputs

Name	Type	Description
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO BYTE	Indicates the address of the string array into which the records are to be written.
cbData	UDINT	Indicates the size of the string array in bytes.
nMaxDocumentSize	UDINT	Indicates the maximum size of a single JSON document from pData.
bDataRelease	BOOL	Releases the cached data.

Return value

Name	Type	Description
ReadAsString	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample:

```
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
```

ReadAsStruct

This method reads a specified number of records from the buffered result data. A structure or an array of a structure is specified in which the data is to be written. The data type schema of this structure should correspond as closely as possible to that of the read data. The variable names are compared with those of the record. A validation makes it possible to detect deviations and respond to them.

If there is a requirement to use different names in the database and in the PLC, the names can be described in the structure with the attribute '*ElementName*' with the assigned name from the database.

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
```

Inputs

Name	Type	Description
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO BYTE	Address of the structure array into which the records are to be written.
cbData	UDINT	Indicates the size of the structure array in bytes.
bValidate	BOOL	Return data are compared with the pData structure array and adjusted if necessary.
pNoSQLValidation	POINTER TO FB_NoSQLValidationEvt	Address of the function block FB_NoSQLValidationEvt that provides further information for validating the call.
bDataRelease	BOOL	Releases the cached data.

Return value

Name	Type	Description
ReadAsStruct	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample:

```
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
```

Release

This method can be used to release data cached by the TwinCAT Database Server.

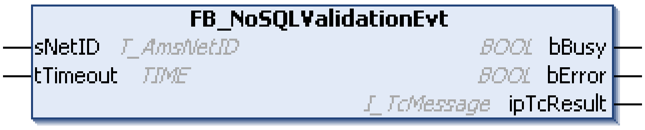
Syntax

```
METHOD Release : BOOL
```

Return value

Name	Type	Description
Release	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

6.1.1.4.4 FB_NoSQLValidationEvt



Function block for reading the validation events and results that occurred when reading the data with FB_NoSQLResultEvt [▶ 199]. This function block is initialized via the CreateValidation method of the NoSQLResult. It refers to the last call of the ReadAsStruct [▶ 201] method.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResult	Tc3_EventLogger.I_TcMessage [▶ 212]	Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▶ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name	Definition location	Description
GetIssues [▶ 204]	Local	Reads a list of validation events as a string array.
GetRemainingData [▶ 204]	Local	Reads the data as a string which could not be assigned to any element in the structure in the PLC.
Release [▶ 205]	Local	Releases the buffered data in the TwinCAT Database Server.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

GetIssues

Reads the validation events that occurred into an array of type T_MAXSTRING. These contain information about remaining or unassigned records or which elements of the PLC structure were not filled.

Syntax

```
METHOD GetIssues : BOOL
VAR_INPUT
    pData : POINTER TO BYTE;
    cbData: UDINT;
    bDataRelease : BOOL;
END_VAR
```

Inputs

Name	Type	Description
pData	POINTER TO BYTE	Address of the array of type T_MAXSTRING in which the records are to be written.
cbData	UDINT	Indicates the size of the string array in bytes.
bDataRelease	BOOL	Releases the cached data.

Return value

Name	Type	Description
ReadAsString	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample:

```
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
```

GetRemainingData

This method can be used to read the remaining data in JSON format after the validation. This includes records which could not be assigned to the PLC structure, for example.

Syntax

```
METHOD GetRemainingData : BOOL
VAR_INPUT
    pData : POINTER TO BYTE;
    cbData : UDINT;
    cbDocument : UDINT;
    bDataRelease : BOOL;
END_VAR
```

Inputs

Name	Type	Description
pData	POINTER TO BYTE	Indicates the address of the string array into which the records are to be written.
cbData	UDINT	Indicates the size of the string array in bytes.
cbDocument	UDINT	Specifies the length of the string in the array.
bDataRelease	BOOL	Releases the cached data.

Return value

Name	Type	Description
GetRemainingData	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample:

```
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
```

Release

Releases the validation results in the memory.

Syntax

```
METHOD Release : BOOL
```

Return value

Name	Type	Description
Release	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

6.1.1.4.5 Helper

These function blocks offer useful functions for dealing with data types, for example.

FB_NoSQLObjectId_MongoDB

The function block for parsing the ObjectId from the MongoDB. In the PLC it is described by the data type T_ObjectId_MongoDB [▸ 222].

Syntax

```
FUNCTION_BLOCK FB_NoSQLObjectId_MongoDB
VAR_INPUT
    ObjectId : T_ObjectId_MongoDB;
END_VAR
```

Inputs

Name	Type	Description
ObjectId	T_ObjectId_MongoDB	12-byte data type for describing the ObjectId.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.
nId	UDINT	Non-unique, sequential number
nMachineId	UDINT	Identification of the machine
nProcessId	UINT	Identification of the writing process
tTimestamp	DATE_AND_TIME	Time stamp of the record

Methods

Name	Definition location	Return value	Description
ToString	Local	STRING(36)	Returns the ID as a string with type designation. Example: ‚ObjectId(„5be15c11afa6ec72b107dafaf")‘
ValueOf	Local	STRING(24)	Returns only the ID as a string. Example: ‚5be15c11afa6ec72b107dafaf‘

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

6.1.1.5 Support of the Tc3_Eventlogger

The TwinCAT 3 Database Server supports the TwinCAT 3 EventLogger (TwinCAT 3 Version 4022.20). This makes it possible to read out details of function block events via an interface. Further information on the EventLogger can be found in the TwinCAT 3 basic libraries.

All function blocks of the TwinCAT 3 Database Server support the interface of the Tc3 EventLogger. The interface Tc3_Eventlogger.I_TcMessage [▸ 212] is used as the return value of the function blocks. In addition to the return value, the *eTraceLevel* property is available to determine the event weighting.

Properties

Name	Type	Description
eTraceLevel	TcEventSeverity [▸ 213]	Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Sample:

As an example, the following weighting is specified for the function block FB_PLCDBWrite:

```
fbPLCDBWriteEvt.eTraceLevel := TcEventSeverity.Warning;
```

All events that represent at least a warning are now sent here. Events of the "Information" weighting are ignored in this case.

The Tc3_Database function blocks themselves have the output *ipTcResult* of data type Tc3_Eventlogger.I_TcMessage. All functions of this interface that are offered can be used.

In this sample, the function block is called first.

```
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
```

If an error occurs, we now want to request the event text in the runtime environment. The *RequestEventText* method can be used for this purpose. Use *nLangId =1031* to read the error code in German. This is one of the many functions of the Tc3_Eventlogger.I_TcMessage [▸ 212] interface.

```
255://Request EventText
    IF myTcMessage.RequestEventText(1031,
        ADR(MyEventString),
        SIZEOF(MyEventString))THEN
        nState := 0;
    END_IF
```

6.1.1.5.1 I_TcEventBase

Methods and properties of an event are defined in this basic interface.

Methods

Name	Description
EqualsTo [▸ 208]	Compares the event with another instance.
EqualsToEventClass [▸ 209]	Compares the event class of the event with another event class.
EqualsToEventEntryEx [▸ 210]	Compares the event definition of the event with another event definition.
GetJsonAttribute [▸ 210]	Returns the Json attribute.
RequestEventClassName [▸ 211]	Requests the name of the event class.
RequestEventText [▸ 212]	Returns the text for the event.

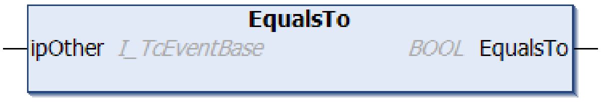
Properties

Name	Type	Access	Description
eSeverity	TcEventSeverity [▸ 213]	Get	Returns the severity.
EventClass	GUID	Get	Returns the GUID of the event class.
ipSourceInfo	I_TcSourceInfo	Get	Returns a pointer to the source definition.
nEventId	UDINT	Get	Returns the ID of the event.
stEventEntry	TcEventEntry	Get	Returns the event definition.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.20	PC or CX (x64, x86, ARM)	Tc3_EventLogger

EqualsTo



This method carries out a comparison with another event specified at the input.

Syntax

```
METHOD EqualsTo : BOOL
VAR_INPUT
    ipOther : I_TcEventBase;
END_VAR
```

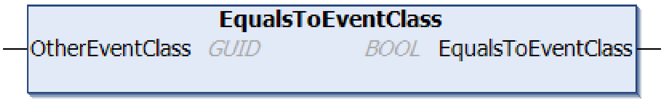
Inputs

Name	Type	Description
ipOther	I_TcEventBase	Event to be compared

Return value

Name	Type	Description
EqualsTo	BOOL	Returns TRUE if the events match.

EqualsToEventClass



This method carries out a comparison with another event class specified at the input.

Syntax

```
METHOD EqualsToEventClass : BOOL
VAR_INPUT
    OtherEventClass : GUID
END_VAR
```

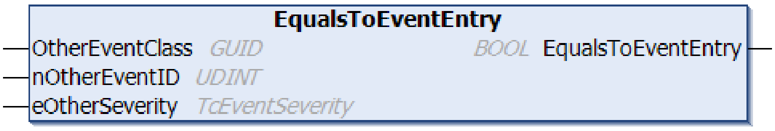
Inputs

Name	Type	Description
OtherEventClass	GUID	Event class to be compared.

Return value

Name	Type	Description
EqualsToEventClass	BOOL	Returns TRUE if the event classes match.

EqualsToEventEntry



This method carries out a comparison with another event specified at the input.

Syntax

```
METHOD EqualsToEventEntry : BOOL
VAR_INPUT
    OtherEventClass : GUID;
    nOtherEventID   : UDINT;
    eOtherSeverity  : TcEventSeverity;
END_VAR
```

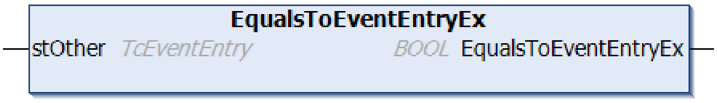
Inputs

Name	Type	Description
OtherEventClass	GUID	Event class of the event to be compared.
nOtherEventID	UDINT	Event ID of the event to be compared.
eOtherSeverity	TcEventSeverity	Event severity of the event to be compared.

Return value

Name	Type	Description
EqualsToEventEntry	BOOL	Returns TRUE if the events match.

EqualsToEventEntryEx



This method carries out a comparison with another event specified at the input.

Syntax

```
METHOD EqualsToEventEntryEx : BOOL
VAR_INPUT
    stOther : TcEventEntry;
END_VAR
```

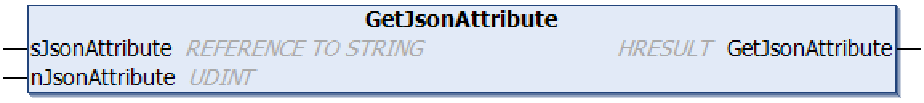
Inputs

Name	Type	Description
stOther	TcEventEntry	Event to be compared.

Return value

Name	Type	Description
EqualsToEventEntryEx	BOOL	Returns TRUE if the events match.

GetJsonAttribute



This method returns the Json attribute.

Syntax

```
METHOD GetJsonAttribute : HRESULT
VAR_INPUT
    sJsonAttribute : REFERENCE TO STRING;
    nJsonAttribute : UDINT;
END_VAR
```

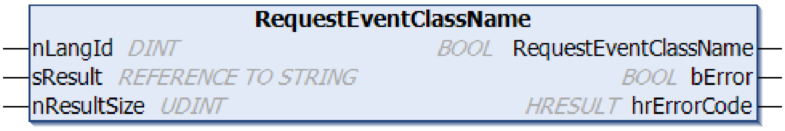
Inputs

Name	Type	Description
sJsonAttribute	REFERENCE TO STRING	Reference to a variable of the type String
nJsonAttribute	UDINT	Length of the String variable

Return value

Name	Type	Description
GetJsonAttribute	HRESULT	Returns S_OK if the method call was successful. Returns ERROR_BAD_LENGTH if the length of the variable is too small. Otherwise HRESULT is returned as error code.

RequestEventClassName



This method returns the name of the event class.

Syntax

```
METHOD RequestEventClassName : BOOL
VAR_INPUT
    nLangId     : DINT;
    sResult     : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

Inputs

Name	Type	Description
nLangId	DINT	Specifies the language ID English (en-US) = 1033 German (de-DE) = 1031 …
sResult	REFERENCE TO STRING	Reference to a variable of the type String
nResultSize	UDINT	Size of the String variable in bytes

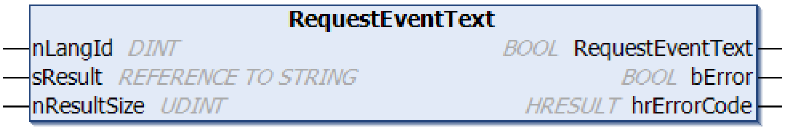
Return value

Name	Type	Description
RequestEventClassName	BOOL	Returns TRUE as soon as the request has been terminated. Returns FALSE if the asynchronous request is still active. The method must be called until the return value is TRUE.

Outputs

Name	Type	Description
bError	BOOL	Returns FALSE if the method call was successful. Returns TRUE if an error has occurred.
hrErrorCode	HRESULT	Returns S_OK if the method call was successful. An error code is output in case of an error.

RequestEventText



This method returns the event text.

Syntax

```
METHOD RequestEventText : BOOL
VAR_INPUT
    nLangId     : DINT;
    sResult     : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError      : BOOL;
    hrErrorCode : HRESULT;
END_VAR
```

Inputs

Name	Type	Description
nLangId	DINT	Specifies the language ID English (en-US) = 1033 German (de-DE) = 1031 …
sResult	REFERENCE TO STRING	Reference to a variable of the type String
nResultSize	UDINT	Size of the String variable in bytes

Return value

Name	Type	Description
RequestEventText	BOOL	Returns TRUE as soon as the request has been terminated. Returns FALSE if the asynchronous request is still active. The method must be called until the return value is TRUE.

Outputs

Name	Type	Description
bError	BOOL	Returns FALSE if the method call was successful. Returns TRUE if an error has occurred.
hrErrorCode	HRESULT	Returns S_OK if the method call was successful. An error code is output in case of an error.

6.1.1.5.2 I_TcMessage

This interface provides methods and properties for the message handling.

Inheritance hierarchy

I_TcEventBase [▶ 207]

I_TcMessage

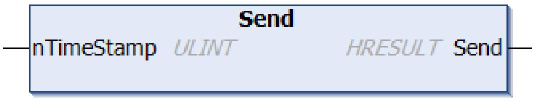
Methods

Name	Description
Send [▸ 213]	Sends a message

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4022.20	PC or CX (x64, x86, ARM)	Tc3_EventLogger

Send



This method sends the message.

Syntax

```
METHOD Send : HRESULT
VAR_INPUT
    nTimeStamp: ULINT;
END_VAR
```

Inputs

Name	Type	Description
nTimeStamp	ULINT	0: Current time stamp is used > 0: External time stamp in 100 nanoseconds since January 1st, 1601 (UTC).

Return value

Name	Type	Description
Send	FB_ HRESULT	Returns S_OK if the method call was successful, otherwise HRESULT as error code

6.1.1.5.3 Data types

TcEventSeverity

Defines the severity of the event.

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE TcEventSeverity : (
    Verbose  := 0,
    Info     := 1,
    Warning  := 2,
    Error    := 3,
    Critical := 4);
END_TYPE
```

6.1.2 Data types

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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.1.5 ST_ConfigAutoLogGrp

This structure is used for the Read method of the function block FB_ConfigTcDBSrvEvt [▶ 145]. All configured AutoLog groups are read into the PLC in an array of this structure.

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	Type	Description
sName	T_MaxString	Group name
hAutoLogGrpID	UDINT	ID of the declared AutoLog group
hDBID	UDINT	ID of the assigned database
sTableName	T_MaxString	Table name
stADSDev	ST_ADSDevice [▶ 230]	ADS device information
eWriteMode	E_WriteMode [▶ 229]	Write mode
nCycleTime	TIME	Cycle time
nSymbolCount	UDINT	Number of symbols

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.1.6 ST_ConfigDB

This structure is used for the Read method of the function block FB_ConfigTcDBSrvEvt [▶ 145]. All configured database connections are read into the PLC in an array of this structure.

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	Type	Parameter
sName	T_MaxString	Connection name
hDBID	DINT	ID of the declared database
eDBType	E_DatabaseType [▶ 214]	Database type
sServer	STRING (80)	Server name
sDatabase	STRING (80)	Database name
bTemp	BOOL	TRUE if the connection was only stored temporarily.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.1.8 ConfigType

Different database configuration structures are provided for the supported database types.

T_DBConfig_ASCII

Describes the database configuration structure for the ASCII file.

Syntax

Definition:

```
TYPE T_DBConfig_ASCII
STRUCT
    sServer: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sServer	T_MaxString	Path to the ASCII file

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_MSAccess

Describes the database configuration structure for a Microsoft Access database.

Syntax

Definition:

```
TYPE T_DBConfig_MSAccess
STRUCT
    sServer: T_MaxString;
    sProvider: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sServer	T_MaxString	Path to the Access databases
sProvider	T_MaxString	Access 2000 – Access 2003: "Microsoft.Jet.OLEDB.4.0" Access 2007: "Microsoft.ACE.OLEDB.12.0"

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_MsCompactSQL

Describes the database configuration structure for a Microsoft Compact SQL database.

Syntax

Definition:

```
TYPE T_DBConfig_MsCompactSQL
STRUCT
    sServer: T_MaxString;
    sPassword: T_MaxString;
    bAuthentification: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sServer	T_MaxString	Path to the Microsoft Compact SQL file (*.sdf)
sPassword	T_MaxString	Password for the database
bAuthentification	BOOL	TRUE if the database is password-protected.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_MsExcel

Describes the database configuration structure for a Microsoft Excel file.

Syntax

Definition:

```
TYPE T_DBConfig_MsExcel
STRUCT
    sServer: T_MaxString;
    sProvider: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sServer	T_MaxString	Path to the Microsoft Excel file
sProvider	T_MaxString	"Microsoft.Jet.OLEDB.4.0" or "Microsoft.ACE.OLEDB.12.0"

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_MsSQL

Describes the database configuration structure for a Microsoft SQL database.

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	Type	Description
sServer	T_MaxString	Enter the name of your SQL server here. Example: "TESTSERVER\SQLEXPRESS"
sProvider	T_MaxString	"SQLOLEDB" or the provider of the SQL native client, e.g. "SQLNCLI10"
Database	T_MaxString	Enter the desired database name here.
sUserID	T_MaxString	Enter the user name here.
sPassword	T_MaxString	Enter the user password here.
bAuthentification	BOOL	Set this variable to TRUE to activate authentication based on user ID and password.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_NET_MySQL

Describes the database configuration structure for a .NET MySQL database.

Syntax

Definition:

```
TYPE T_DBConfig_NET_MySQL
STRUCT
    sServer: T_MaxString;
    sDatabase: T_MaxString;
    nPort: UDINT;
    sUserID: T_MaxString;
    sPassword: T_MaxString;
    bAuthentification: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sServer	T_MaxString	Enter the name or IP address of your server here.
Database	T_MaxString	Enter the desired database name here.
nPort	UDINT	Enter the port for communicating with the MySQL database here. Default: 3306.
sUserID	T_MaxString	Enter the user name.
sPassword	T_MaxString	Enter the user password here.
bAuthentification	BOOL	Set this variable to TRUE to activate authentication based on user ID and password.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_Odbc

Describes the database configuration structure for a database with ODBC interface.

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	Type	Description
eOdbcSubType	E_OdbcSubType [▶ 215]	Describes an ODBC database [▶ 215] that is supported with full functionality.
nParameterCount	UINT	Number of parameters for the connection strings
arrParameter	ARRAY [0..MAX_CONFIGPARAMETER] OF ST_ConnStringParameter [▶ 217];	Array of parameters for the connection string of type ST_ConnStringParameter [▶ 217].

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_SQLite

Describes the database configuration structure for an SQLite database.

Syntax

Definition:

```
TYPE T_DBConfig_SQLite
STRUCT
    sServer: T_MaxString;
    sPassword: T_MaxString;
    bAuthentification: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sServer	T_MaxString	Path to the SQLite file
sPassword	T_MaxString	Password for the database
bAuthentification	BOOL	TRUE if the database is password-protected.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

T_DBConfig_XML

Describes the database configuration structure for an XML file in the customized database format.

Syntax

Definition:

```
TYPE T_DBConfig_XML
STRUCT
    sServer: T_MaxString;
    sSchema: T_MaxString;
    sDatabase: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sServer	T_MaxString	Name and path of the XML file
sSchema	T_MaxString	Name and path of the XSD file
sDatabase	T_MaxString	Describes the name of the database. The XML, XSD and XSL files are created automatically for this database type.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
);
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.2.3 T_ObjectId_MongoDB

A data type that maps the database-specific "ObjectId" data type. It consists of 12 bytes with different meanings:

Table 1: ObjectId at byte level

1	2	3	4	5	6	7	8	9	10	11	12
Timestamp				MachineId			ProcessId		Id		

The function block FB_NoSQLObjecId_MongoDB [▶ 206] is available for parsing out the individual elements.

Syntax

```
TYPE T_ObjectId_MongoDB : ARRAY[0..11] OF BYTE;
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.2.4 QueryOptions

DocumentDB

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	Type	Description
pFilter	POINTER TO BYTE	Specifies the address of the search filter based on which the collection is to be searched.
cbFilter	UDINT	Length of the search filter.
pSort	POINTER TO BYTE	Specifies the sort address based on which the collection is to be sorted.
cbSort	UDINT	Length of sorting
pProjection	POINTER TO BYTE	Specifies the address of the display and how the collection data is to be displayed.
cbProjection	UDINT	Length of the display.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

T_QueryOptionDocumentDB_Aggregate

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Aggregate
STRUCT
    pPipeStages: POINTER TO BYTE;
    cbPipeStages: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
pPipeStages	POINTER TO BYTE	Specifies the address of the stage document. Several stages can be defined in this.
cbPipeStages	UDINT	Length of the stage document.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

T_QueryOptionDocumentDB_Insert

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Insert
STRUCT
    pDocuments: POINTER TO BYTE;
    cbDocuments: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
pDocuments	POINTER TO BYTE	Specifies the address of one or more documents to add to a collection.
cbDocuments	UDINT	Length of the document.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

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	Type	Description
pFilter	POINTER TO BYTE	Specifies the address of the search filter based on which the collection is to be searched.
cbFilter	UDINT	Length of the search filter.
pDocuments	POINTER TO BYTE	Specifies the address of the documents whose values are to be transferred to the collection.
cbDocuments	UDINT	Length of the documents

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or CX (x86)	Tc3_Database

T_QueryOptionDocumentDB_Delete

Syntax

Definition:

```
TYPE T_QueryOptionDocumentDB_Delete
STRUCT
    pFilter: POINTER TO BYTE;
    cbFilter: UDINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
pFilter	POINTER TO BYTE	Specifies the address of the search filter based on which the collection is to be searched.
cbFilter	UDINT	Length of the search filter.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1 Build 4022.20	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Values

Name	Description
check_None	PLC structure is not verified.
check_OK	No differences between PLC structure and database table structure
check_TypeWarning	Different data types with regard to the sign between PLC and database data types, e.g. UINT <> INT
check_TypeError	Different data types with regard to the sign between PLC structure and database table structure
check_DataLengthWarning	Different length between PLC structure and database table structure. The record can nevertheless be mapped, although data may be lost.
check_DataLengthError	Different length between PLC structure and database table structure. The record cannot be mapped.

6.1.2.3.4 ST_SQLSPParameter

This structure is required for the function block FB_SQLStoredProcedureEvt [▶ 191] to describe the different parameters of the procedure to be executed.

Syntax

Definition:

```
TYPE ST_SQLSPParameter :
STRUCT
    eParameterType: E_SPParameterType;
    eParameterDataType: E_ColumnType;
    nParameterSize: UDINT;
    sParameterName: STRING(50);
END_STRUCT
END_TYPE
```

Name	Type	Description
eParameterType	E_SPParameterType [▶ 226]	Parameter type (INPUT, OUTPUT ..)
eParameterDataType	E_ColumnType [▶ 225]	Parameter type
nParameterSize	UDINT	Parameter length
sParameterName	STRING (50)	Parameter name

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.4.8 ST_ADSDevice

Describes the ADS device, which has to be specified for methods of the function block FB_PLCDBWriteEvt [▶ 168].

Syntax

Definition:

```
TYPE ST_ADSDevice :
STRUCT
    sDevNetID: T_AmsNetId;
    nDevPort: T_AmsPort;
    eADSRdWrtMode: E_ADSRdWrtMode;
    tTimeout: TIME;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sDevNetID	T_AmsNetId	NetID of the ADS device
nDevPort	T_AmsPort	AMS Port
eADSRdWrtMode	E_ADSRdWrtMode [▶ 227]	Connection mode IGroup_IOffset / bySymbol
tTimeout	TIME	ADS connection timeout

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.4.9 ST_AutoLogGrpStatus

Provides information about the respective AutoLog group.

Syntax

Definition:

```
TYPE ST_AutoLogGrpStatus :
STRUCT
    hAutoLogGrpID: UDINT;
    nCycleCount: UDINT;
    hrErrorCode: HRESULT;
    eErrorType: E_ErrorType;
    bError: BOOL;
END_STRUCT
END_TYPE
```

Parameter

Parameter

Name	Type	Description
hAutoLogGrpID	UDINT	ID of the declared AutoLog group
nCycleCount	UDINT	Number of executed cycles
hrErrorCode	HRESULT	HRESULT error code
eErrorType	E_ErrorType [▶ 227]	Error type
bError	BOOL	TRUE if an error has occurred.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or 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	Type	Description
sName	STRING (50)	Name of the column
sProperty	STRING	String for additional column properties
nLength	UDINT	Maximum length (for strings and byte streams)
eType	E_ColumnType [▶ 225]	Column type

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.4.11 ST_ExpParameter

This structure is required for the function block FB_PLCCmd [▶ 173], for making the description of the different parameters (placeholders) available in the SQL command.

Syntax

Definition:

```
TYPE ST_ExpParameter:
STRUCT
    sParaName : T_MaxString;
    nParaSize : UDINT;
    eParaType : E_ExpParameterType;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sParaName	T_MaxString	Name of the parameter (placeholder)
nParaSize	UDINT	Length of the parameter value
eParaType	E_ExpParameterType [▶ 228]	Data type of the parameter

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.4.12 ST_StandardRecord

This structure can be used in the PLC if you want to work with the standard table structure of the TwinCAT Database Server.

This structure cannot be used with Microsoft Access databases, since this database type does not support the 64-bit integer data type. In this case the structure ST_StandardRecord_MSAccess [▸ 232] should be used.

Syntax

Definition:

```
TYPE ST_StandardRecord :
STRUCT
    nID: LINT;
    dtTimestamp: DT;
    sName: STRING(80);
    rValue: LREAL;
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.4.13 ST_StandardRecord_MSAccess

This structure can be used in the PLC if you want to work with the standard table structure of the TwinCAT Database Server. This structure is specifically intended for Microsoft Access databases, since this database type does not support the 64-bit integer data type.

Syntax

Definition:

```
TYPE ST_StandardRecord_MSAccess:
STRUCT
    nID: DINT;
    dtTimestamp: DT;
    sName: STRING(80);
    rValue: LREAL;
END_STRUCT
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.2.4.14 ST_Symbol

Describes the ADS symbol, which has to be specified for methods of the function block FB_PLCDBWriteEvt [▸ 168].

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	Type	Description
sSymbolName	T_MaxString	Symbol name
sDBSymbolName	T_MaxString	Name to be written to the database.
nIGroup	UDINT	Index Group (only for ADSRdWrtMode "eADSMode_IGroup_IOffset")
nIOffset	UDINT	Index Offset (only for ADSRdWrtMode "eADSMode_IGroup_IOffset")
nBitSize	UDINT	Length in bits (only for ADSRdWrtMode "eADSMode_IGroup_IOffset")
eDataType	E_PLCDataType [▶ 229]	Data type (only for ADSRdWrtMode "eADSMode_IGroup_IOffset")

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.3 Global constants

6.1.3.1 Constants

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;
```

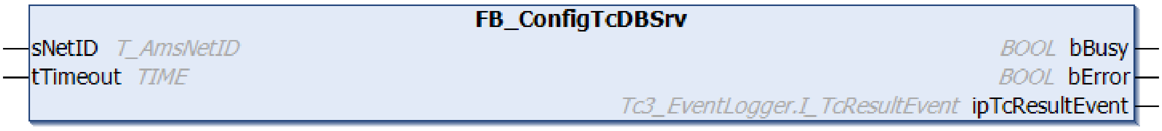
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

6.1.4 Obsolete

6.1.4.1 Configure mode

6.1.4.1.1 FB_ConfigTcDBSrv



Function block for creating, reading and deleting configuration entries for the 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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Create [▶ 146]	Local	Creates new entries in the XML configuration file for the TwinCAT Database Server
Read [▶ 147]	Local	Reads the current configuration of the TwinCAT Database Server
Delete [▶ 148]	Local	Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Create

This method creates new entries in the XML configuration file for the TwinCAT Database Server. Optionally the TwinCAT Database Server can use a new entry on a temporary basis. In this case no data is written to the XML file.

Syntax

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

Inputs

Name	Type	Description
pTcDBSrvConfig	POINTER TO BYTE	Pointer of the configuration structure to be created.
cbTcDBSrvConfig	UDINT	Length of the configuration structure
bTemporary	BOOL	Indicates whether the configuration is to be stored in the XML file.
pConfigID	POINTER TO UDINT	Return pointer of the configuration ID (hDBID or hAutoLogGrpID)

Creating AutoLog groups is currently not supported.

Return value

Name	Type	Description
Create	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Read

This method can be used to read the current configurations of the TwinCAT Database Server. Any temporary configurations that may be included are marked accordingly.

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
```

Inputs

Name	Type	Description
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB [▶ 216]	Pointer address of the array into which the database configurations are to be written.
cbDBConfig	UDINT	Length of the database configuration array
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp [▶ 215]	Pointer address of the array into which the AutoLogGrp configurations are to be written.
cbAutoLogGrpConfig	UDINT	Length of the AutoLogGrp configuration array
pDBCount	POINTER TO UDINT	Pointer address for storing the number of database configurations.
pAutoLogGrpCount	POINTER TO UDINT	Pointer address for storing the number of AutoLogGrp configurations.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Delete

This method can be used to delete databases and AutoLog groups from the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
eTcDBSrvConfigType	E_TcDBSrvConfigType	Type of the configuration to be deleted (database / AutoLog group)
hConfigID	UDINT	ID of the configuration to be deleted (hDBID or hAutoLogGrpID)

Return value

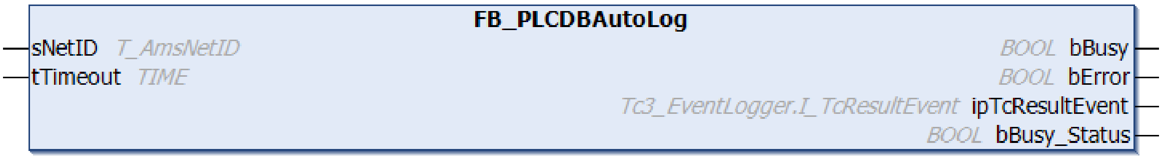
Name	Type	Description
Delete	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block with four methods for starting and stopping of defined AutoLog groups and for reading of the corresponding group status.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active, except for the Status method.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.
bBusy_Status	BOOL	The Status method can be executed independently of the other three methods of the function block and therefore has its own Busy flag. Is TRUE as soon as the Status method is active.

Methods

Name	Definition location	Description
RunOnce [▶ 150]	Local	Executes the AutoLog group once
Start [▶ 151]	Local	Starts AutoLog mode with the corresponding configured AutoLog groups
Status [▶ 151]	Local	Queries the status of the AutoLog groups.
Stop [▶ 152]	Local	Stops AutoLog mode

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

RunOnce

This method can be used to execute an AutoLog group once, for example based on an event in the controller.

Syntax

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

Inputs

Name	Type	Description
hAutoLogGrpID	UDINT	ID of the AutoLog group to be executed once.
bAll	BOOL	If TRUE, all AutoLog groups are executed once.

Return value

Name	Type	Description
RunOnce	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Start

This method starts the AutoLog mode with the corresponding configured AutoLog groups.

Syntax

```
METHOD Start : BOOL
```

Return value

Name	Type	Description
Start	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Status

This method can be used to query the status of the AutoLog groups. A separate busy flag is provided in the body of the function block for this method, since it can be called independently of the other methods of the function block: 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
```

Inputs

Name	Type	Description
tCheckCycle	TIME	Interval time at which the status array is updated.
pError	POINTER TO BOOL	TRUE, if an error has occurred in AutoLog mode.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus	Address of the status array that contains all groups.
cbAutoLogStatus	UDINT	Length of the status array

Return value

Name	Type	Description
Status	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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(aAutologG
rpStatus)) THEN
    ; // ...
END_IF
```

Stop

This method stops the AutoLog mode.

Syntax

```
METHOD Stop : BOOL
```

Return value

Name	Type	Description
Stop	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

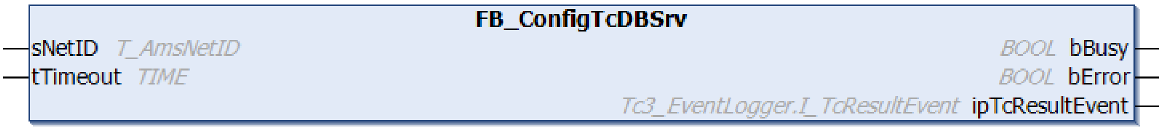
Sample

```
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



Function block for creating, reading and deleting configuration entries for the 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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Create [▸ 242]	Local	Creates new entries in the XML configuration file for the TwinCAT Database Server
Read [▸ 243]	Local	Reads the current configuration of the TwinCAT Database Server
Delete [▸ 244]	Local	Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Create

This method creates new entries in the XML configuration file for the TwinCAT Database Server. Optionally the TwinCAT Database Server can use a new entry on a temporary basis. In this case no data is written to the XML file.

Syntax

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

Inputs

Name	Type	Description
pTcDBSrvConfig	POINTER TO BYTE	Pointer of the configuration structure to be created.
cbTcDBSrvConfig	UDINT	Length of the configuration structure
bTemporary	BOOL	Indicates whether the configuration is to be stored in the XML file.
pConfigID	POINTER TO UDINT	Return pointer of the configuration ID (hDBID or hAutoLogGrpID)

Creating AutoLog groups is currently not supported.

Return value

Name	Type	Description
Create	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Read

This method can be used to read the current configurations of the TwinCAT Database Server. Any temporary configurations that may be included are marked accordingly.

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
```

Inputs

Name	Type	Description
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB [▶ 216]	Pointer address of the array into which the database configurations are to be written.
cbDBConfig	UDINT	Length of the database configuration array
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp [▶ 215]	Pointer address of the array into which the AutoLogGrp configurations are to be written.
cbAutoLogGrpConfig	UDINT	Length of the AutoLogGrp configuration array
pDBCount	POINTER TO UDINT	Pointer address for storing the number of database configurations.
pAutoLogGrpCount	POINTER TO UDINT	Pointer address for storing the number of AutoLogGrp configurations.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Delete

This method can be used to delete databases and AutoLog groups from the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
eTcDBSrvConfigType	E_TcDBSrvConfigType	Type of the configuration to be deleted (database / AutoLog group)
hConfigID	UDINT	ID of the configuration to be deleted (hDBID or hAutoLogGrpID)

Return value

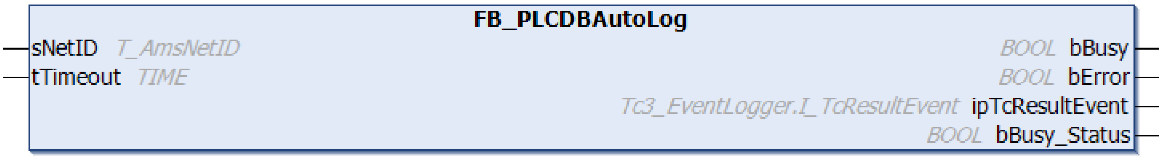
Name	Type	Description
Delete	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block with four methods for starting and stopping of defined AutoLog groups and for reading of the corresponding group status.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active, except for the Status method.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.
bBusy_Status	BOOL	The Status method can be executed independently of the other three methods of the function block and therefore has its own Busy flag. Is TRUE as soon as the Status method is active.

Methods

Name	Definition location	Description
RunOnce [▶ 150]	Local	Executes the AutoLog group once
Start [▶ 151]	Local	Starts AutoLog mode with the corresponding configured AutoLog groups
Status [▶ 151]	Local	Queries the status of the AutoLog groups.
Stop [▶ 152]	Local	Stops AutoLog mode

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

RunOnce

This method can be used to execute an AutoLog group once, for example based on an event in the controller.

Syntax

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

Inputs

Name	Type	Description
hAutoLogGrpID	UDINT	ID of the AutoLog group to be executed once.
bAll	BOOL	If TRUE, all AutoLog groups are executed once.

Return value

Name	Type	Description
RunOnce	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Start

This method starts the AutoLog mode with the corresponding configured AutoLog groups.

Syntax

```
METHOD Start : BOOL
```

Return value

Name	Type	Description
Start	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

Status

This method can be used to query the status of the AutoLog groups. A separate busy flag is provided in the body of the function block for this method, since it can be called independently of the other methods of the function block: 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
```

Inputs

Name	Type	Description
tCheckCycle	TIME	Interval time at which the status array is updated.
pError	POINTER TO BOOL	TRUE, if an error has occurred in AutoLog mode.
pAutoLogStatus	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus	Address of the status array that contains all groups.
cbAutoLogStatus	UDINT	Length of the status array

Return value

Name	Type	Description
Status	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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(aAutologG
rpStatus)) THEN
    ; // ...
END_IF
```

Stop

This method stops the AutoLog mode.

Syntax

```
METHOD Stop : BOOL
```

Return value

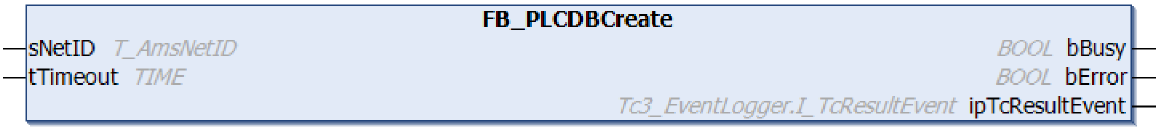
Name	Type	Description
Stop	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

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

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

6.1.4.2.3 FB_PLCDBCreate



Function block with two methods. One method can be used to create databases from the PLC on a database server specified in the PLC. The other method can be used to generate a new table in a specified database.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Database [▸ 248]	Local	Creates a new database
Table [▸ 249]	Local	Creates a new table with a structure that is defined via an array with x elements or x columns in the PLC.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Database

This method creates a new database. Optionally you can specify whether the created database should also be used for the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
pDatabaseConfig	POINTER TO BYTE	Address of the database configuration structure [▶ 217]
cbDatabaseConfig	UDINT	Length of the database configuration structure
bCreateXMLConfig	BOOL	Indicates whether the newly created database should be entered as new configuration entry in the XML file.
pDBID	UDINT	Returns the hDBID if/when a new configuration entry was created.

Return value

Name	Type	Description
Database	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Table

This method creates a new table with a structure that is defined through an array with x elements or x columns in the PLC.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	MaxString	Name of the table to be created.
pTableCfg	POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ColumnInfo [▶ 231]	Indicates the pointer address of the table structure array. The individual columns are written in this array.
cbTableCfg	UDINT	Indicates the length of the array in which the columns are configured.

Return value

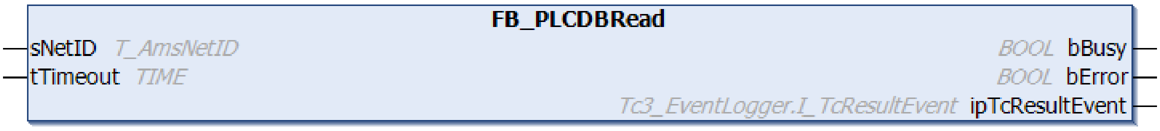
Name	Type	Description
Table	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block for reading records from a database.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Read [▶ 251]	Local	Reads a specified number of records from a database table with the standard table structure specified by Beckhoff.
ReadStruct [▶ 253]	Local	Reads a specified number of records from a database table with any table structure.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Read

This method reads a specified number of records from a database table with the standard table structure specified by Beckhoff. (The standard table structure is used in AutoLog mode and in the FB_DBWrite function block, for example).

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
sDBSymbolName	T_MaxString	Symbol name to be read from the standard table structure.
eOrderBy	E_OrderColumn.eColumnID	Sorting column (ID, timestamp, name or value)
eOrderType	E_OrderType.eOrder_ASC	Sorting direction (ASC or DESC)
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO ST_StandardRecord	Address of the structure array into which the records are to be written.
cbData	UDINT	Indicates the size of the structure array in bytes.

Return value

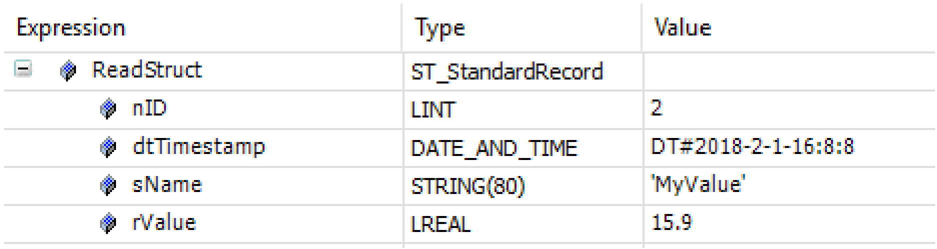
Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Result in the PLC:



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

ReadStruct

This method reads a specified number of records from a database table with any table structure.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Address of the array containing the column name to be read.
cbColumnNames	UDINT	Length of the column name array
sOrderByColumn	STRING(50)	Name the sorting column
eOrderType	E_OrderType	Sorting direction (ASC or DESC)
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO BYTE	Address of the structure array into which the records are to be written.
cbData	UDINT	Indicates the size of the structure array in bytes.

Return value

Name	Type	Description
ReadStruct	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

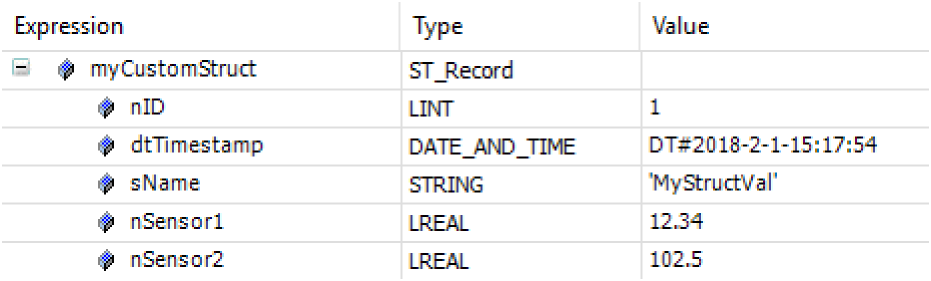
Sample

```
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
```

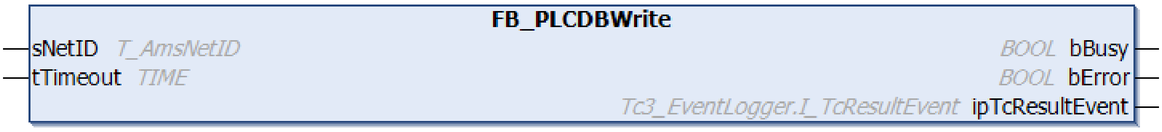
```
// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';
ColumnNames[3] := 'Sensor1';
ColumnNames[4] := 'Sensor2';

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
```

Result in the PLC:



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



Function block for writing of records into a database.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Write [▶ 255]	Local	Creates a record in the standard table structure specified by Beckhoff.
WriteBySymbol [▶ 256]	Local	Reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff.
WriteStruct [▶ 258]	Local	Creates a record with any table structure

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Write

This method creates a record in the standard table structure specified by Beckhoff.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
pValue	POINTER TO BYTE	Address of the variable to be logged in the standard table structure.
cbValue	UDINT	Length of the variable to be logged.
sDBSymbolName	T_MaxString	Name that is logged in the table.
eDBWriteMode	E_WriteMode	Indicates the write mode. (append, update, ring buffer)
nRingBuffParameter	UDINT	Additional parameter(s) for the "ring buffer" write mode.

Return value

Name	Type	Description
Write	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

This sample shows how to use the FB_PLCDBWriteEvt.Write method:

```
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
```

Result in the database:

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

With the ring buffer option, only three entries of this name are in the database at any one time. Older entries are deleted.

WriteBySymbol

This method reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff. ADS symbols from other ADS devices can also be read.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
stADSDevice	ST_ADSDevice	ADS device from which a symbol is to be logged in the standard table structure.
stSymbol	ST_Symbol	Symbol name of the variable to be written
eDBWriteMode	E_WriteMode	Indicates the write mode. (append, update, ring buffer)
nRingBuffParameter	UDINT	Additional parameter(s) for the "ring buffer" write mode

Return value

Name	Type	Description
WriteBySymbol	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

This sample shows how to use the FB_PLCDBWriteEvt.WriteBySymbol method:

```
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
```

Result in the database:

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

WriteStruct

This method creates a record with a freely selectable table structure.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
sTableName	T_MaxString	Name of the table that is to be read.
pRecord	POINTER TO BYTE	Address of a structure that is to be logged in a freely selectable table structure.
cbRecord	UDINT	Length of the structure to be written
pColumnNames	POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)	Address of the array containing the column name to be filled.
cbColumnNames	UDINT	Length of the column name array

Return value

Name	Type	Description
WriteStruct	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

This sample shows how to use the method 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
```

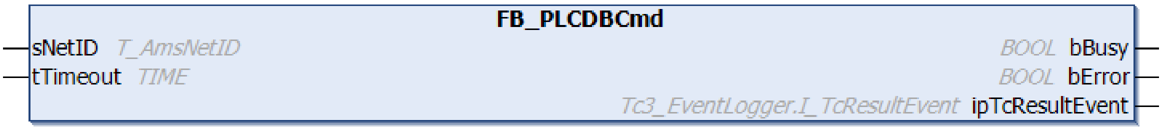
```
TYPE ST_Record :
STRUCT
    nID         : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName       : STRING;
    nSensor1    : LREAL;
    nSensor2    : LREAL;
END_STRUCT
END_TYPE
```

```
// 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
```

Result in the database:

ID	Timestamp	Name	Sensor1	Sensor2
5	'2018-01-30 15:23:26'	‘MyStructVal’	12.34	102.5

6.1.4.2.6 FB_PLCDBCmd



Function block with two methods. Users can define and transfer their own SQL commands. Placeholders in the SQL command can correlate with structures in the PLC, which reflect the table structure. The Database Server ultimately enters the current data of the structure into the SQL command.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Execute [▸ 260]	Local	Sends any SQL commands to the database. Returned records cannot be read.
ExecuteDataReturn [▸ 262]	Local	Sends any SQL commands to the database. A specified number of records can be read.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Execute

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are then replaced by the TwinCAT Database Server with the corresponding values before the execution. Returned records cannot be read.

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
```

Sample

```
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
```

ExecuteDataReturn

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are then replaced by the TwinCAT Database Server with the corresponding values before the execution. A specified number of records can be read.

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
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.
pExpression	POINTER TO BYTE	Address of the string variable with the SQL command.
cbExpression	UDINT	Length of the string variable with the SQL command.
pData	POINTER TO BYTE	Address of the structure with the parameter values
cbData	UDINT	Length of the structure with the parameter values
pParameter	POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter	Address of the structure array with the parameter information.
cbParameter	UDINT	Length of the structure array with the parameter information.
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pReturnData	POINTER TO BYTE	Address of the structure array into which the records are to be written.
cbReturnData	UDINT	Indicates the size of the structure array in bytes.
pRecords	POINTER TO BYTE	Number of read records.

Return value

Name	Type	Description
ExecuteDataReturn	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Parameterizing the command

The column names for the individual parameters are specified in curly brackets in the SQL command.
Sample: ‚SELECT * FROM MyHouse_Temperatures WHERE Room = {SelectedRoom}’.
Accordingly, SelectedRoom has to be specified as parameter name in the structure ST_ExpParameter.

Some databases do not support the parameterization of SQL clauses. (TOP/LIMIT/ROWNUM/...)
Parameterizable table names are not usually supported.

Sample

```
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

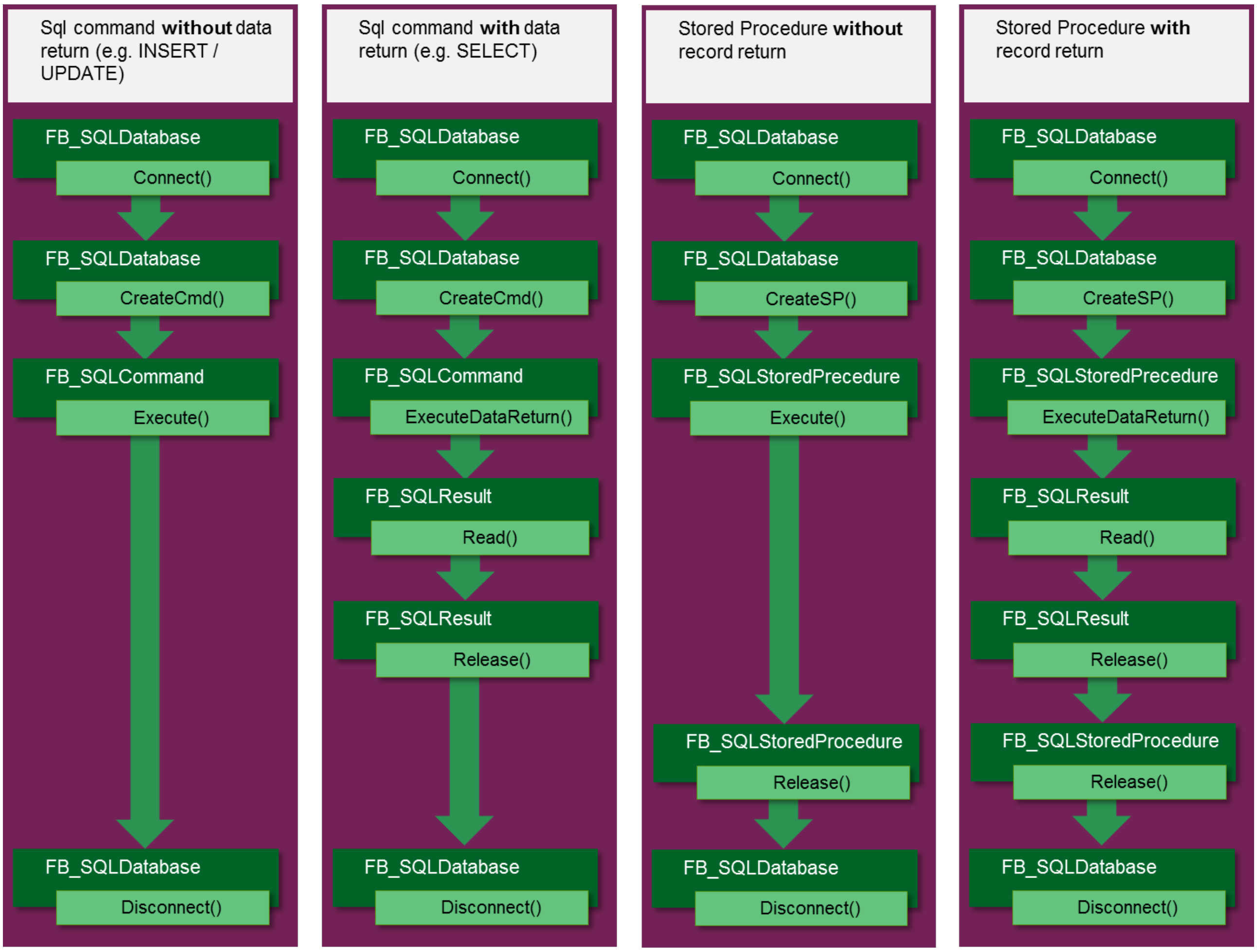
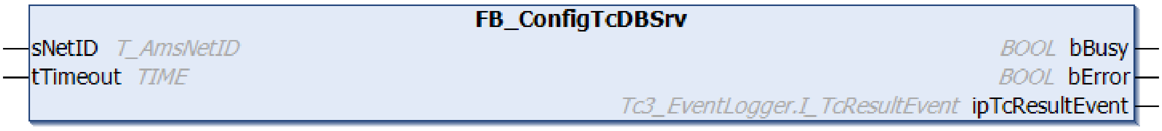


Fig. 2:

6.1.4.3.1 FB_ConfigTcDBSrv



Function block for creating, reading and deleting configuration entries for the 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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Create [▶ 146]	Local	Creates new entries in the XML configuration file for the TwinCAT Database Server
Read [▶ 147]	Local	Reads the current configuration of the TwinCAT Database Server
Delete [▶ 148]	Local	Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Create

This method creates new entries in the XML configuration file for the TwinCAT Database Server. Optionally the TwinCAT Database Server can use a new entry on a temporary basis. In this case no data is written to the XML file.

Syntax

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

Inputs

Name	Type	Description
pTcDBSrvConfig	POINTER TO BYTE	Pointer of the configuration structure to be created.
cbTcDBSrvConfig	UDINT	Length of the configuration structure
bTemporary	BOOL	Indicates whether the configuration is to be stored in the XML file.
pConfigID	POINTER TO UDINT	Return pointer of the configuration ID (hDBID or hAutoLogGrpID)

Creating AutoLog groups is currently not supported.

Return value

Name	Type	Description
Create	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Read

This method can be used to read the current configurations of the TwinCAT Database Server. Any temporary configurations that may be included are marked accordingly.

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
```

Inputs

Name	Type	Description
pDBConfig	POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB [▶ 216]	Pointer address of the array into which the database configurations are to be written.
cbDBConfig	UDINT	Length of the database configuration array
pAutoLogGrpConfig	POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp [▶ 215]	Pointer address of the array into which the AutoLogGrp configurations are to be written.
cbAutoLogGrpConfig	UDINT	Length of the AutoLogGrp configuration array
pDBCount	POINTER TO UDINT	Pointer address for storing the number of database configurations.
pAutoLogGrpCount	POINTER TO UDINT	Pointer address for storing the number of AutoLogGrp configurations.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Delete

This method can be used to delete databases and AutoLog groups from the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

Name	Type	Description
eTcDBSrvConfigType	E_TcDBSrvConfigType	Type of the configuration to be deleted (database / AutoLog group)
hConfigID	UDINT	ID of the configuration to be deleted (hDBID or hAutoLogGrpID)

Return value

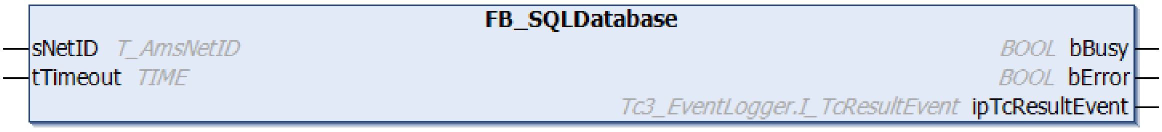
Name	Type	Description
Delete	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block for opening, closing and managing a database connection.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Connect [▶ 269]	Local	Opens a connection to a declared database.
CreateCmd [▶ 270]	Local	Initializes an instance of the function block FB_SQLCommand with the already open database connection of the function block FB_SQLDatabase.
CreateSP [▶ 270]	Local	Initializes an instance of the function block FB_SQLStoredProcedure with the already open database connection of the function block FB_SQLDatabase.
Disconnect [▶ 271]	Local	Closes the connection to the database that was opened by this function block instance.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Connect

This method opens a connection to a declared database.

Syntax

```
METHOD Connect : BOOL
VAR_INPUT
    hDBID: UDINT := 1;
END_VAR
```

Inputs

Name	Type	Description
hDBID	UDINT	Indicates the ID of the database to be used.

Return value

Name	Type	Description
Connect	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

CreateCmd

This method is used to initialize an instance of the function block FB_SQLCommand with the already open database connection of the function block FB_SQLDatabase. The function block FB_SQLCommand only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLCommand can be initialized with the same database connection.

The initialization of the function block FB_SQLCommand is completed in the same cycle. This means that neither the Busy flag of the function block nor the method return value of the CreateCmd method have to be checked.

Syntax

```
METHOD CreateCmd : BOOL
VAR_INPUT
    pSQLCommand: POINTER TO FB_SQLCommand;
END_VAR
```

Inputs

Name	Type	Description
pSQLCommand	POINTER TO FB_SQLCommand	Returns a new instance of the function block FB_SQLCommand.

Return value

Name	Type	Description
CreateCmd	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

FB_SQLCommandEvt [▶ 185] can then be used for the execution.

CreateSP

This method is used to initialize an instance of the function block FB_SQLStoredProcedure with the already open database connection of the function block FB_SQLDatabase. The function block FB_SQLStoredProcedure only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLStoredProcedure can be initialized with the same database connection.

The initialization of the function block FB_SQLStoredProcedure may take several cycles. The Busy flag of the function block or the method return value of the CreateCmd method have to be checked before the function block can be used.

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
```

Inputs

Name	Type	Description
sProcedureName	T_MaxString	Indicates the name of the procedure to be executed.
pParameterInfo	POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter	Pointer address for the parameter info list.
cbParameterInfo	UDINT	Indicates the length of the parameter info list.
pSQLProcedure	POINTER TO FB_SQLStoredProcedure	Returns a new instance of the function block FB_SQLStoredProcedure.

Return value

Name	Type	Description
CreateSP	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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
```

Subsequently, the FB_SQLStoredProcedureEvt [▸ 191] can be used to execute the stored procedure.

Disconnect

This method closes the connection to the database that was opened by this function block instance.

Syntax

```
METHOD Disconnect : BOOL
```

Return value

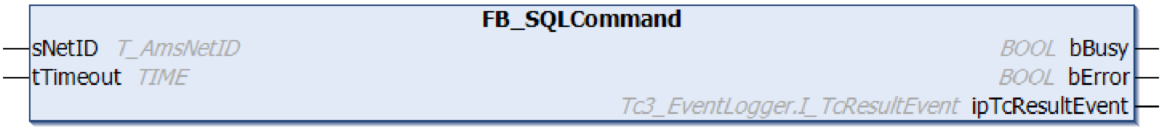
Name	Type	Description
Disconnect	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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



Function block for executing SQL commands. Before it can be used it has to be initialized with the function block FB_SQLDatabase.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Execute [▶ 273]	Local	Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase.
ExecuteDataReturn [▶ 274]	Local	Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the function block FB_SQLResult can be transferred for reading the returned records.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Execute

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
END_VAR
```

Inputs

Name	Type	Description
pSQLCmd	POINTER TO BYTE	Indicates the pointer address of a string variable with the SQL command to be executed.
cbSQLCmd	UDINT	Indicates the length of a SQL command to be executed.

Return value

Name	Type	Description
Execute	POINTER TO BYTE	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

Uses the command created by FB_SQLDatabaseEvt.CreateCmd() [▶ 181].

```
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
```

ExecuteDataReturn

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the function block FB_SQLResult can be transferred for reading the returned records.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
    pSQLDBResult: POINTER TO FB_SQLResult;
END_VAR
```

Inputs

Name	Type	Description
pSQLCmd	POINTER TO BYTE	Indicates the pointer address of a string variable with the SQL command to be executed.
cbSQLCmd	UDINT	Indicates the length of a SQL command to be executed.
pSQLDBResult	POINTER TO FB_SQLResult [▶ 275]	Returns a new instance of the function block FB_SQLResult.

Return value

Name	Type	Description
ExecuteDataReturn	POINTER TO BYTE	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

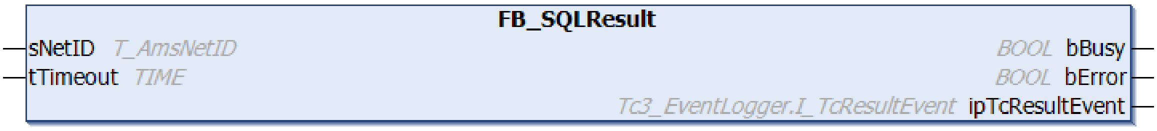
Uses the command created by FB_SQLDatabaseEvt.CreateCmd() [▶ 181].

```
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
```

FB_SQLResultEvt [▶ 188] can then be used to read the data.

6.1.4.3.4 FB_SQLResult



The function block is used for reading the cached records.

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Read [▸ 275]	Local	Reads a specified number of records from the result data cached in the TwinCAT Database Server.
Release [▸ 277]	Local	Releases data buffered by the TwinCAT Database Server.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Read

This method reads a specified number of records from the result data cached in the TwinCAT Database Server.

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
```

Inputs

Name	Type	Description
nStartIndex	UDINT	Indicates the index of the first record to be read.
nRecordCount	UDINT	Indicates the number of records to be read.
pData	POINTER TO BYTE	Address of the structure array into which the records are to be written.
cbData	UDINT	Indicates the size of the structure array in bytes.
bWithVerifying	BOOL	Return data are compared with the pData structure array and adjusted if necessary.
bDataRelease	BOOL	Releases the cached data.

Return value

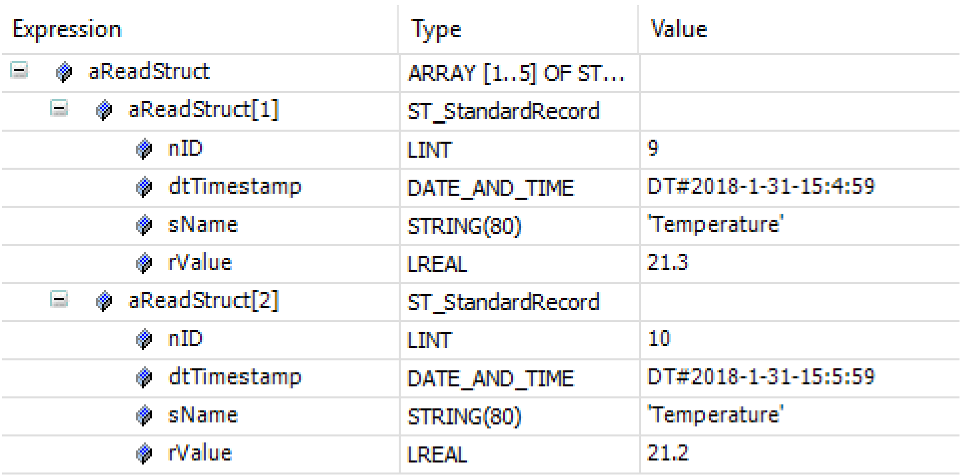
Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

```
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), TRUE, TRUE) THEN
    IF fbSqlResult.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

Result in the PLC:



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

Release

This method can be used to release data cached by the TwinCAT Database Server.

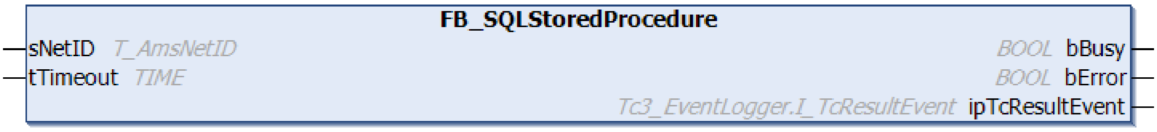
Syntax

```
METHOD Release : BOOL
```

Return value

Name	Type	Description
Release	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

6.1.4.3.5 FB_SQLStoredProcedure



Function block for executing stored procedures of the database. Before it can be used it has to be initialized with the function block "FB_SQLDatabase".

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
```

Inputs

Name	Type	Description
sNetID	T_AmsNetID	AMS network ID of the target device at which the ADS command is directed.
tTimeout	TIME	Indicates the time before the function is cancelled.

Outputs

Name	Type	Description
bBusy	BOOL	TRUE as soon as a method of the function block is active.
bError	BOOL	TRUE when an error occurs.
ipTcResultEvent	Tc3_EventLogger.I_TcResultEvent	Result interface with detailed information on the return value.

Methods

Name	Definition location	Description
Execute [▸ 278]	Local	Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase.
ExecuteDataReturn [▸ 279]	Local	Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the FB_SQLResult function block can be transferred for reading the returned records.
Release [▸ 279]	Local	Releases the parameter information of the stored procedure that was transferred during initialization.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.1 Build 4020.10	PC or CX (x86)	Tc3_Database

Execute

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase.

Syntax

```
METHOD Execute : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
END_VAR
```

Inputs

Name	Type	Description
pParameterStrc	POINTER TO BYTE	Pointer address to the parameter structure that is transferred to the procedure.
cbParameterStrc	UDINT	Length of the parameter structure

Return value

Name	Type	Description
Execute	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [▸ 181].

```
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
```

ExecuteDataReturn

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the FB_SQLResult function block can be transferred for reading the returned records.

Syntax

```
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
    pSQLDBResult: POINTER TO FB_SQLDBResult;
END_VAR
```

Inputs

Name	Type	Description
pParameterStrc	POINTER TO BYTE	Pointer address to the parameter structure that is transferred to the procedure.
cbParameterStrc	UDINT	Length of the parameter structure
pSQLDBResult	POINTER TO FB_SQLDBResult	Returns a new instance of the function block FB_SQLDBResult.

Return value

Name	Type	Description
Read	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [▶ 181].

```
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
```

FB_SQLResultEvt [▶ 188] can then be used to read the data.

Release

This method releases the parameter information of the stored procedure, which was transferred during initialization.

Syntax

```
METHOD Release : BOOL
```

Return value

Name	Type	Description
Release	BOOL	Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.

6.2 Tc2_Database

Overview

The Tc2_Database library contains function blocks for controlling and configuring the TwinCAT 3 database server.

Function blocks

Name	Description
FB_GetStateTcDatabase [▶ 282]	Retrieves status information.
FB_DBConnectionAdd [▶ 284]	Adds database connections to the XML configuration file.
FB_DBAuthentificationAdd [▶ 303]	Adds authentication information for the respective database connection to the XML configuration file.
FB_DBOdbcConnectionAdd [▶ 285]	Adds an ODBC database connection to the XML configuration file.
FB_AdsDeviceConnectionAdd [▶ 287]	Adds an ADS device to the XML configuration file.
FB_DBReloadConfig [▶ 283]	Reloads the XML configuration file
FB_GetDBXMLConfig [▶ 288]	Reads all database configurations from the XML configuration file.
FB_GetAdsDevXMLConfig [▶ 288]	Reads all ADS device configurations from the XML configuration file.
FB_DBConnectionOpen [▶ 289]	Opens a connection to a database.
FB_DBConnectionClose [▶ 290]	Closes a connection to a database.
FB_DBCreate [▶ 291]	Creates a new database
FB_DBTableCreate [▶ 292]	Creates a table with any desired table structure
FB_DBRead [▶ 294]	Reads one value out of the database
FB_DBWrite [▶ 295]	Writes one variable value, with timestamp, into a database
FB_DBCyclicRdWrt [▶ 293]	Starts or stops the logging/writing of variables
FB_DBRecordSelect [▶ 305]	Reads a data record out of a table
FB_DBRecordSelect_EX [▶ 307]	Reads a data record out of a table (command length less than 10,000 characters)
FB_DBRecordArraySelect [▶ 298]	Reads several records from a table.
FB_DBRecordInsert [▶ 304]	Creates a new data record.
FB_DBRecordInsert_EX [▶ 297]	Creates a new data record. (command length less than 10,000 characters)
FB_DBRecordDelete [▶ 296]	Deletes a record from a table.
FB_DBStoredProcedures [▶ 301]	Executes a stored procedure.
FB_DBStoredProceduresRecordReturn [▶ 308]	Executes a stored procedure and returns a record.
FB_DBStoredProceduresRecordArray [▶ 302]	Executes a stored procedure and returns several records.

Data Types

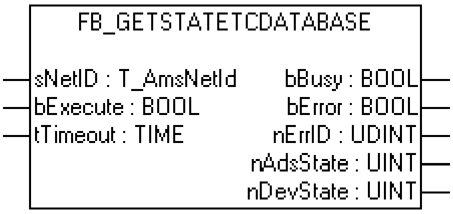
Name
ST_DBColumnCfg [▸ 309]
ST_DBXMLCfg [▸ 309]
ST_ADSDevXMLCfg [▸ 310]
ST_DBSQLError [▸ 310]
ST_DBParameter [▸ 311]
E_DbColumnTypes [▸ 311]
E_DBTypes [▸ 312]
E_DBValueType [▸ 312]
E_DBWriteModes [▸ 312]
E_DBParameterTypes [▸ 313]

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1 Function blocks

6.2.1.1 FB_GetStateTcDatabase



The function block allows to get the current state of the Twincat Database Server.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

bExecute: The command is executed with the rising edge.

tTimeout: Indicates the timeout time.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy     : BOOL;
    bError    : BOOL;
    nErrID    : UDINT;
    nAdsSta   : UINT;
    nDevState : UINT;
END_VAR
```

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code if the bError output is set.

nAdsState: Contains the state identification code of the ADS target device. The codes returned here are specified for all ADS servers:

- 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;

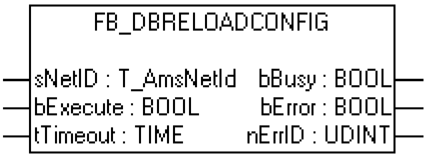
nDevState: Contains the specific state identification code of the ADS target device. The codes returned here are supplementary information specific to the ADS device.

- 1 = TwinCAT Database Server started
- 2 = cyclic reading or writing started

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.2 FB_DBReloadConfig



With the FB_DBReloadConfig function block the XML configuration file can be reloaded.
If the XML configuration file was modified, the Database Server must be notified of the modifications with the aid of FB_DBReloadConfig.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

bExecute: The command is executed with a rising edge.

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

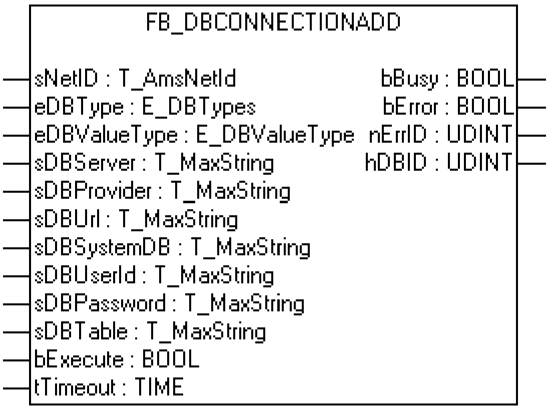
bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code if the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.3 FB_DBConnectionAdd



The FB_DBConnectionAdd function block permits additional database connections to be added to the XML configuration file.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

eDBType: Indicates the type of the database, e.g. 'Mobile server'.

eDBValueType: Indicates the form in which the values are or will be stored.

sDBServer: Provides the name of the server: Optional.

sDBProvider: Gives the provider of the database: Optional.

sDBUrl: Gives the path to the database.

sSystemDB: Only for Access databases. Indicates the path to the MDW file.

sUserId: Indicates the login user name.

sPassword: Indicates the password.

sDBTable: Gives the name of the table into which the values are to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

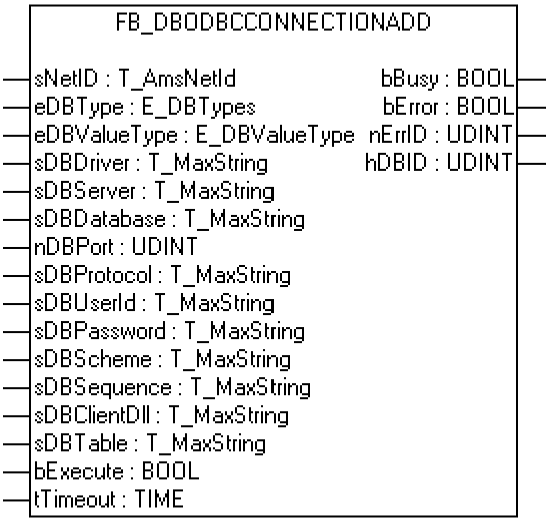
nErrID: Returns the ADS error code if the bError output is set.

hDBID: Returns the ID of the database.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.4 FB_DBOdbcConnectionAdd



The function block FB_DBOdbcConnectionAdd can be used to add further ODBC database connections to the XML configuration file.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

eDBType: Indicates the type of the database, e.g. 'Mobile server'.

eDBValueType: Indicates the form in which the values are or will be stored.

sDBDriver: Indicates the name of the ODBC driver to be used.

sDBServer: Indicates the name of the server.

sDBDatabase: Indicates the name of the database.

nDBPort: Indicates the port for the ODBC connection.

sDBProtocol: Indicates the protocol to be used (TCPIP).

sDBUserId: Indicates the user name.

sDBPassword: Indicates the password to be used.

sDBScheme: Indicates the database schema to be used.

sDBSequence: Indicates the sequence name for Oracle databases.

sDBClientDll: Contains the path to fbclient.dll. (Only for Firebird/Interbase databases)

sDBTable: Gives the name of the table into which the values are to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

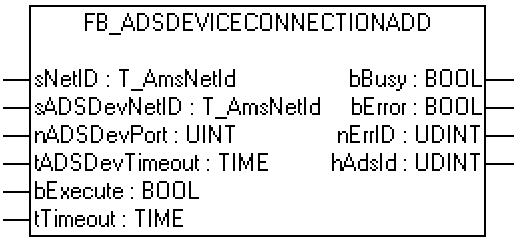
nErrID: Returns the ADS error code if the bError output is set.

hDBID: Returns the ID of the database.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.5 FB_AdsDeviceConnectionAdd



The function block FB_AdsDeviceConnectionAdd permits additional Ads-Device connections to be added to the XML configuration file.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

sADSDevNetID: String containing the AMS network ID of the ADS device.

nADSDevPort: Indicates the port of the ADS device.

tAdsDevTimeout: Indicates the timeout time of the ADS device.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the duration of the timeout.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

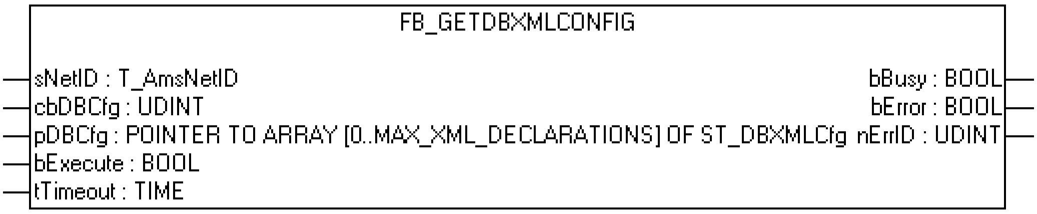
nErrID: Returns the ADS error code if the bError output is set.

hAdsId: Returns the ID of the ADS device.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.6 FB_GetDBXMLConfig



With this function block FB_GetDBXMLConfig all declared databases can be read out of the XML-configuration file.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

cbDBCfg: Indicates the length of the array, into which the configurations are to be written.

pDBCfg: Indicates the pointer address of the array, into which the configurations are to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

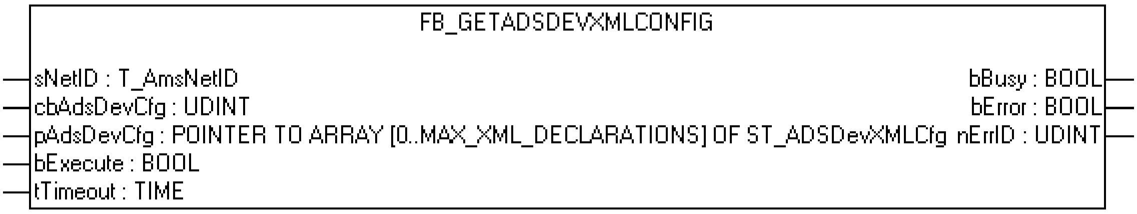
bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code if the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.7 FB_GetAdsDevXMLConfig



With this function block FB_GetAdsDevXMLConfig all declared ADS-devices can be read out of the XML-configuration file.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

cbAdsDevCfg: Indicates the length of the array, into which the configurations are to be written.

pAdsDevCfg: Indicates the pointer address of the array, into which the configurations are to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

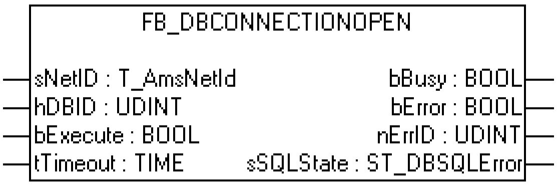
bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code if the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.8 FB_DBConnectionOpen



You can open connections to databases with this function block FB_DBConnectionOpen. This can improve the read and write access speed with the fuction blocks FB_DBWrite, FB_DBRead, FB_DBRecordInsert and FB_FBRecordSelect.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

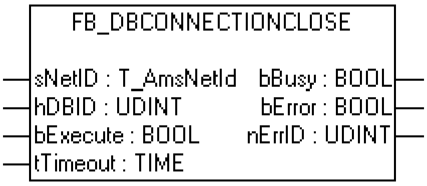
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState : Returns the SQL error code [▶ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.9 FB_DBConnectionClose



The function block FB_DBConnectionClose can be used to make connections with databases. If a connection with a database was opened previously, it must be closed again.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

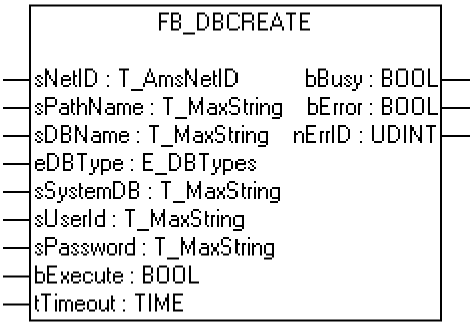
bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code if the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.10 FB_DBCreate



The FB_DBCreate function block allows databases to be created.

The following database types can be created with this function block: MS SQL databases, MS SQL Compact databases, MS Access databases and XML databases

ASCII files can (but do not have to) be created with the function block FB_DBCreate. If they do not exist, they are created automatically during the first write access. They only have to be declared in the XML configuration file.

It is not possible to create DB2, Oracle, MySQL, PostgreSQL, InterBase and Firebird databases. In addition, it is not possible to overwrite existing databases. In this case the function block FB_DBCreate would return an error.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

sPathName: Gives the path to the database.

sDBName: Gives the name of the database that is to be created.

eDBType: Gives the type of the database that is to be created.

sSystemDB: Only for Access databases. Contains the path to the MDW file.

sUserID: User name for the corresponding registration

sPassword: Corresponding password

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the duration of the timeout.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code if the bError output is set.

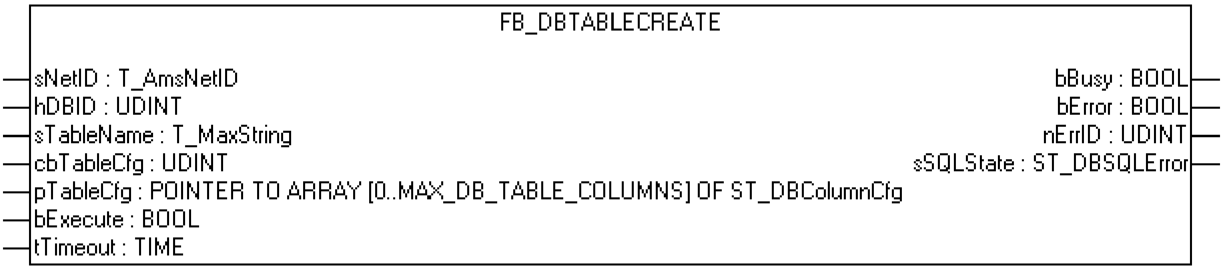
TwinCAT Database Server

If the newly created databases are to be used by the TwinCAT Database Server, the connection data have to be written to the XML configuration file with the aid of the function block FB_DBConnectionADD.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.11 FB_DBTableCreate



The FB_DBTableCreate function block permits tables with any desired table structure to be created in databases.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: ID of the database to be used.

sTableName: Provides the name of the table.

cbTableCfg: Returns the length of the array in which the columns are configured.

pTableCfg: Provides the pointer address of the table structure array. The individual columns are written in this array.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the duration of the timeout.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

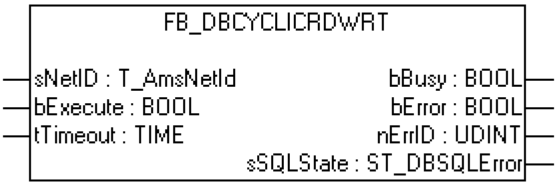
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▸ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▸ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.12 FB_DBCyclicRdWrt



The FB_DBCyclicRdWrt function block can be used to start or stop the cyclic logging \ writing of variables.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

bExecute: The read/write cycle is started with a rising edge and stopped with a falling edge.

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

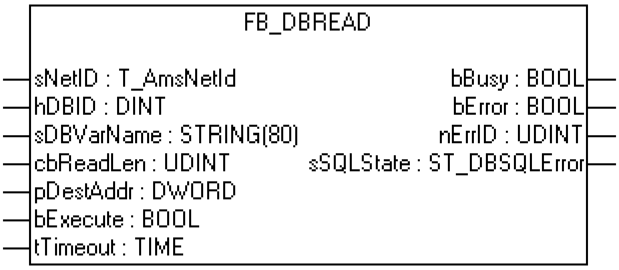
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▶ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.13 FB_DBRead



The FB_DBRead allows values to be read from a database.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

sDBVarName: Gives the name of the variable that is to be read.

cbReadLen: Indicates the length of the buffer that is to be read.

pDestAddr: Contains the address of the buffer which is to receive the data that has been read.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

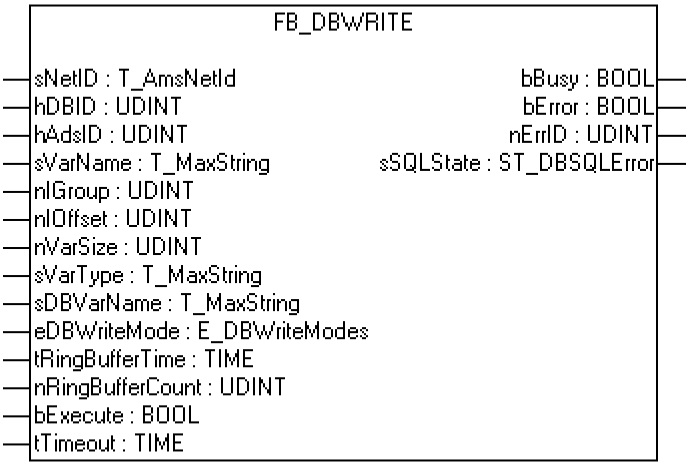
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▸ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.14 FB_DBWrite



The FB_DBWrite function block can be used to write the values of individual variables into databases. The table structure must contain the columns "Timestamp", "Name", and "Value" (see "SQL Compact database" [▸ 122]). In order to be able to use the function block, the database that is to be used for write access and the ADS device, from which the variables are to be read, must be declared in the XML configuration file.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: ID of the database to be used.

hAdsID: ID of the ADS device to be used.

sVarName: Provides the name of the variable.

nIGroup: Index group of the variable (optional, only on the BC9000).

nIOffset: Index offset of the variable (optional, only on the BC9000).

nVarSize: Size of the variable in bytes (optional, only on the BC9000).

sVarType: Data type of the variable (optional, only on the BC9000).

Possible variable data types: "BOOL" / "LREAL" / "REAL" / "INT16" / "DINT" / "USINT" / "BYTE" / "UDINT" / "DWORD" / "UINT16" / "WORD" / "SINT"
sDBVarName: Variable name to be used in the database.

eDBWriteMode: Indicates whether the values are to be appended in new records or whether the existing records are to be updated.

tRingBufferTime: Indicates the maximum age of records in a table (only for Ringbuffer_WriteMode).

nRingBufferCount: Indicates the maximum number of records in a table (only for Ringbuffer_WriteMode).

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

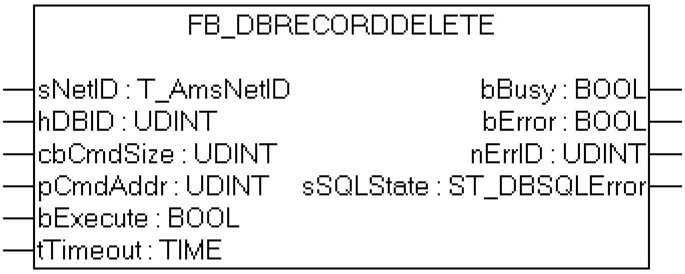
sSQLState: Returns the SQL error code [▶ 310] of the corresponding database type

Log values of ADS devices (except BC9000)	Log values of 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);

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.15 FB_DBRecordDelete



The function block FB_DBRecordDelete can be used to delete individual records from a database. This function block can be used to execute SQL DELETE commands with up to 10,000 characters.
To use the function block it is necessary to declare the database from which the records are to be deleted in the XML configuration file.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

cbCmdSize: Indicates the length of the INSERT command.

pCmdAddr: Pointer to the executing INSERT command.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

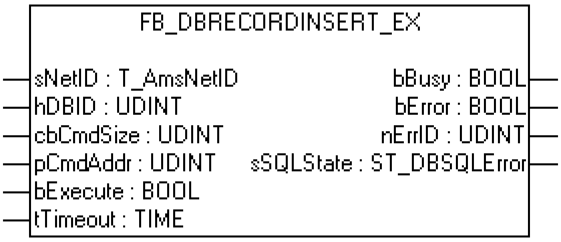
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▶ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.16 FB_DBRecordInsert_EX



The function block FB_DBRecordInsert_EX can be used to write individual records with any structure into a database. This function block can be used to execute SQL INSERT commands with up to 10,000 characters.
To use the function block it is necessary to declare the database to which the records are to be written in the XML configuration file.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

cbCmdSize: Indicates the length of the INSERT command.

pCmdAddr: Pointer to the executing INSERT command

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

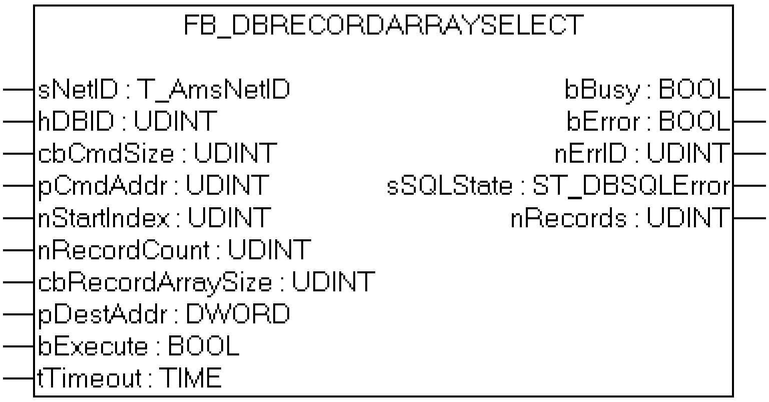
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▸ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▸ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.17 FB_DBRecordArraySelect



The function block FB_DBRecordArraySelect can be used to read several records with any structure from the database. This function block can be used to execute an SQL SELECT command with up to 10,000 characters.
This function block is not compatible with 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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

cbCmdSize: Indicates the length of a SELECT command to be executed.

pCmdSize: Indicates the pointer address of a string variable with the SQL command to be executed.

nStartIndex: Indicates the index of the first record to be read.

nRecordCount: Indicates the number of records to be read.

cbRecordArraySize: Indicates the size of the structure array in bytes.

pDestAddr: Indicates the address of the structure array into which the records are to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

ST_DBSQLError [▶ 310]

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code of the corresponding database type

nRecords: Returns the number of data records.

Sample in ST

Since the table, from which the records are to be read, has the structure below, a PLC structure with a similar structure must be created.

Table:

Column name	Data type
ID	Bigint
Timestamp	datetime
Name	nvarchar(80)
Value	float

Structure:

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

The library TcUtilities.lib must be integrated in order to be able to use the data type T_ULARGE_INTEGER.

For ARM processors the data types have to be arranged differently due to the byte alignment, and a "dummy byte" has to be added.

```
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
```

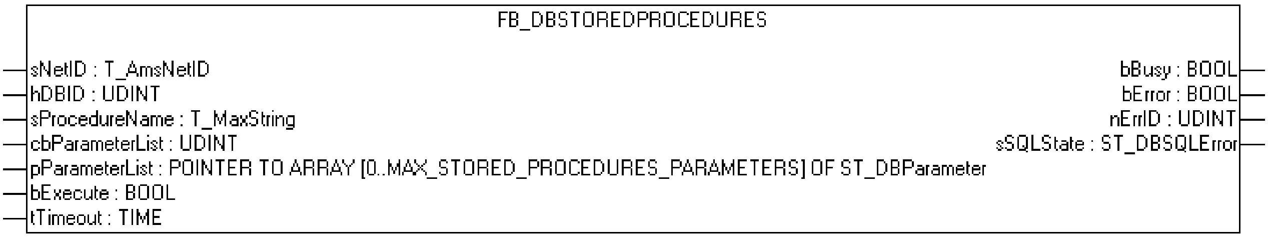
PLC program

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

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.18 FB_DBStoredProcedures



The function block FB_DBStoredProcedures can be used to call up stored procedures. They can include parameters in the process, which are used in the stored procedures.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

sProcedureName: Indicates the name of the procedure to be executed

cbParameterList: Indicates the length of the parameter list.

pParameterList: Contains the address of the parameter list

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

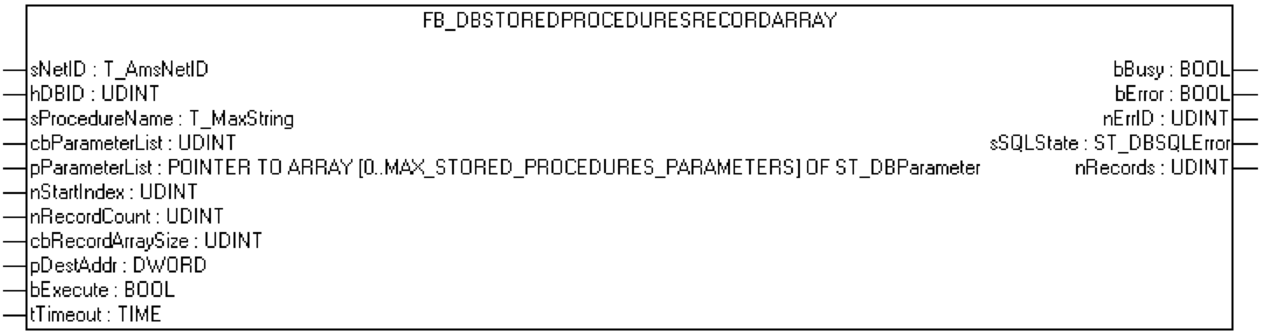
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [⏵ 385] if the bError output is set.

sSQLState: Returns the SQL error code [⏵ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.19 FB_DBStoredProceduresRecordArray


FB_DBSTOREDPROCEDURESRECORDARRAY
sNetID : T_AmsNetID
hDBID : UDINT
sProcedureName : T_MaxString
cbParameterList : UDINT
pParameterList : POINTER TO ARRAY [0..MAX_STORED_PROCEDURES_PARAMETERS] OF ST_DBParameter
nStartIndex : UDINT
nRecordCount : UDINT
cbRecordArraySize : UDINT
pDestAddr : DWORD
bExecute : BOOL
tTimeout : TIME
bBusy : BOOL
bError : BOOL
nErrID : UDINT
sSQLState : ST_DBSQLError
nRecords : UDINT

The function block FB_DBStoredProceduresRecordArray can be used to call stored procedures that return records. In contrast to the FB_DBStoredProceduresRecordReturn function block, this function block can be used to return several records with a single call. They can include parameters in the process, which are used in the stored procedures.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

sProcedureName: Indicates the name of the procedure to be executed.

cbParameterList: Indicates the length of the parameter list.

pParameterList: Contains the address of the parameter list

nStartIndex: Indicates the index of the first record to be read.

nRecordCount: Indicates the number of records to be read.

cbRecordArraySize: Indicates the size of the structure array in bytes.

pDestAddr: Indicates the address of the structure array into which the records are to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as "bBusy" remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▶ 310] of the corresponding database type

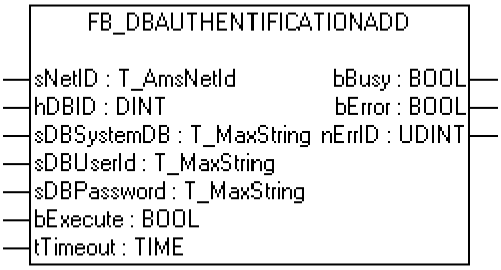
nRecords: Returns the number of data records.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.20 Obsolete

6.2.1.20.1 FB_DBAuthentificationAdd



The function block FB_DBAuthentificationAdd permits authentication information of declared database connection to be added to the XML configuration file or to be changed.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Is the ID of the database to be used.

sSystemDB: Only for Access databases. Indicates the path to the MDW file.

sUserId: Indicates the login user name.

sPassword: Indicates the password.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the duration of the timeout.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as "bBusy" remains TRUE.

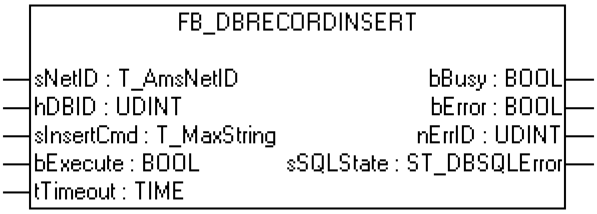
bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code if the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.20.2 FB_DBRecordInsert



The function block FB_DBRecordInsert can be used to write individual records with any structure into a database. To use the function block it is necessary to declare the database to which the records are to be written in the XML configuration file.

VAR_INPUT

```
VAR_INPUT
    sNetID    : T_AmsNetId;
    hDBID     : UDINT;
    sInsertCmd: T_MaxString;
    bExecute  : BOOL;
    tTimeout  : TIME;
END_VAR
```

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

sInsertCmd: Indicates which INSERT command is to be executed.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

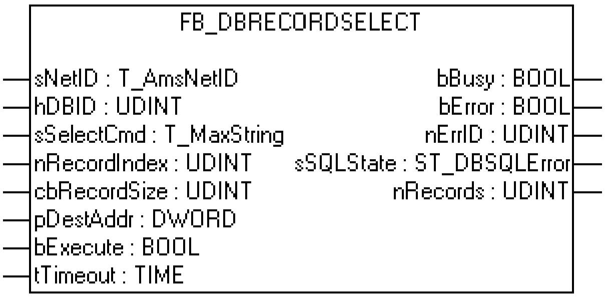
nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▶ 310] of the corresponding database type

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.20.3 FB_DBRecordSelect



The FB_DBRecordSelect allows individual data records to be read from a database.
This function block is not compatible with 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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

sSelectCmd: Indicates which SELECT command is to be executed.

nRecordIndex: Gives the index of the data record that is to be read.

cbRecordSize: Provides the size of a data record in bytes.

pDestAddr: Indicates the address of the structure to which the record is to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▸ 310] of the corresponding database type

nRecords: Returns the number of data records.

Example in ST:

Since the table, from which the records are to be read, has the structure below, a PLC structure with a similar structure must be created.

Table:

Column name	Data type
ID	Bigint
Timestamp	datetime
Name	Ntext
Value	Float

Structure:

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

The library TcUtilities.lib must be integrated in order to be able to use the data type T_ULARGE_INTEGER.

For ARM processors the data types have to be arranged differently due to the byte alignment, and a "dummy BYTE" has to be added.

```
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
```

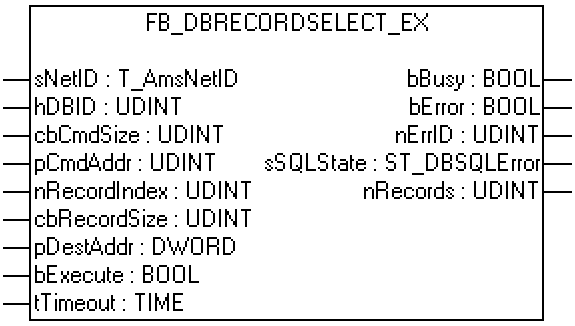
PLC program

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

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.20.4 FB_DBRecordSelect_EX



The function block FB_DBRecordSelect_EX can be used to read individual records with any structure from the database. This function block can be used to execute an SQL SELECT command with up to 10,000 characters.
This function block is not compatible with 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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

cbCmdSize: Indicates the length of a SELECT command to be executed.

pCmdSize: Indicates the pointer address of a string variable with the SQL command to be executed.

nRecordIndex: Gives the index of the data record that is to be read.

cbRecordSize: Provides the size of a data record in bytes.

pDestAddr: Indicates the address of the structure to which the record is to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

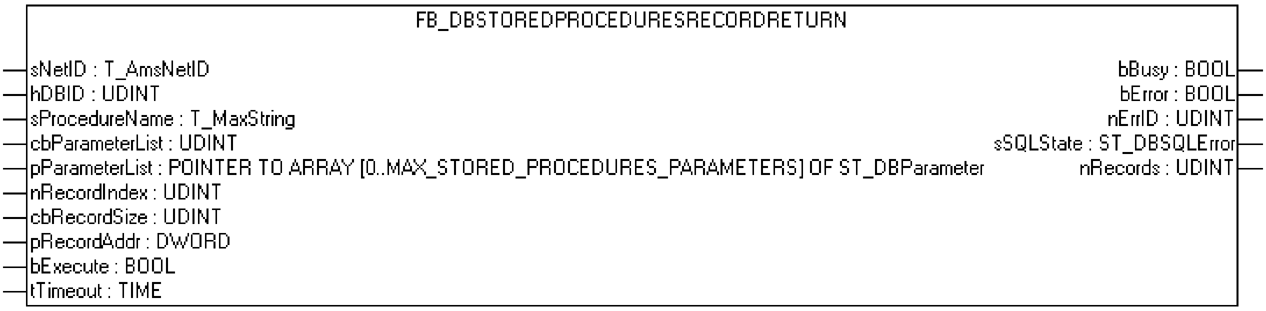
sSQLState: Returns the SQL error code [▶ 310] of the corresponding database type

nRecords: Returns the number of data records.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.1.20.5 FB_DBStoredProceduresRecordReturn



The function block FB_DBStoredProceduresRecordReturn can be used to call up stored procedures that return a record. They can include parameters in the process, which are used in the stored procedures.

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 containing the AMS network ID of the target device, at which the ADS command is directed.

hDBID: Indicates the ID of the database to be used.

sProcedureName: Indicates the name of the procedure to be executed.

cbParameterList: Indicates the length of the parameter list.

pParameterList: Contains the address of the parameter list

nRecordIndex: Gives the index of the data record that is to be read.

cbRecordSize: Provides the size of a data record in bytes.

pRecordAddr: Indicates the address of the structure to which the record is to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [▶ 385] if the bError output is set.

sSQLState: Returns the SQL error code [▶ 310] of the corresponding database type

nRecords: Returns the number of data records.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.2 Data types

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: Contains the name of the column to be created.

sColumnProperty: Contains certain column properties.

eColumnType: Gives the type of column.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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: Contains the name of the database.

sDBTable: Contains the name of the table.

nDBID: Returns the ID of the database.

eDBType: Gives the type of database.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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 containing the AMS network ID of the ADS device.

tAdsDevTimeout: Indicates the timeout time of the ADS device.

nAdsDevID: Returns the ID of the ADS device.

nAdsDevPort: Indicates the port of the ADS device.

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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: Contains the 5-character error code, which is based on the SQL ANSI standard.

nSQLErrorCode: Returns a database-specific error code.

If no error has occurred, the structure contains the following values:
sSQLState := **'00000'**;
nSQLErrorCode := **0**;

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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: Indicates the name of the parameter.

cbParameterValue: Contains the size of the variable to be used in bytes.

pParameterValue: Contains the address of the variable to be used.

eParameterDataType: Indicates the data type of the parameter (E_DBColumnTypes [▸ 311]).

eParameterType: Indicates the parameter type (E_DBParameterTypes [▸ 313]).

Declaration sample

Variable Declaration

```
PROGRAM MAIN
VAR
    paraList: ARRAY [0..2] OF ST_DBParameter;
    p1: DINT := 3;
    p2: LREAL;
    p3: STRING;
END_VAR
```

PLC program

```
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);
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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,
    eDBType_NET_MySQL       := 12
);
END_TYPE
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.2.8 E_DBValueType

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

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or 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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.3 Global Constants

6.2.3.1 Constants

```
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
```

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

6.2.3.2 Library version

All libraries have a certain version. The version is indicated in the PLC library repository, for example. A global constant contains the information about the library version:

Global_Version

```
VAR_GLOBAL CONSTANT
    stLibVersion_TC3_Database_Server : ST_LibVersion;
END_VAR
```

Use the function F_CmpLibVersion (defined in the Tc2_System library) to check whether you are using the correct version.

All other options for comparing library versions, which you may know from TwinCAT 2, are outdated!

7 Examples

7.1 Tc3_Database

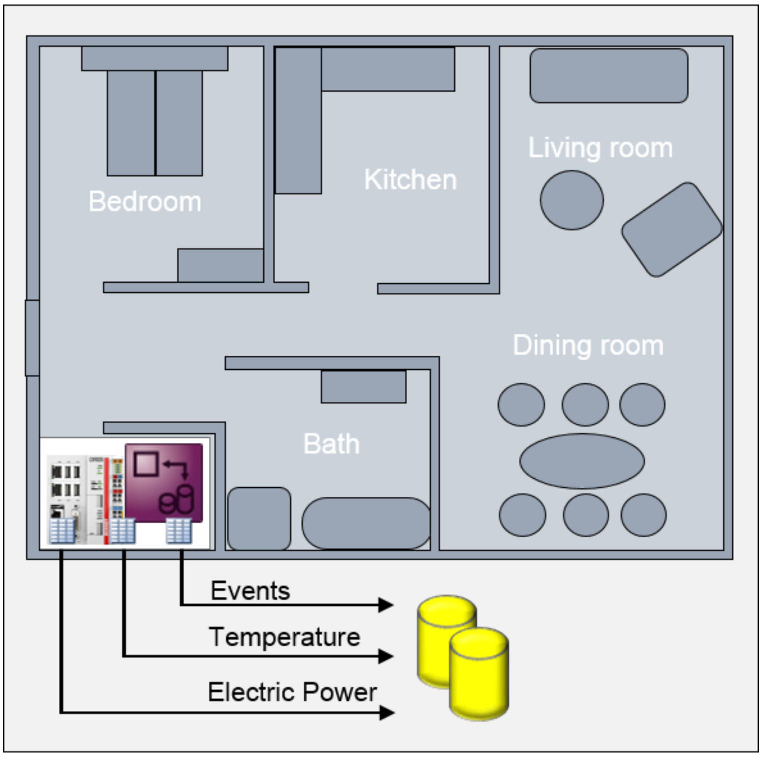
The following pages introduce samples and tips for using the TwinCAT Database Server with the library for TwinCAT 3.

7.1.1 Scenario examples

Each sample is assigned to a scenario, which can be adapted to specific use cases. Information on which databases the samples are compatible with is also provided.

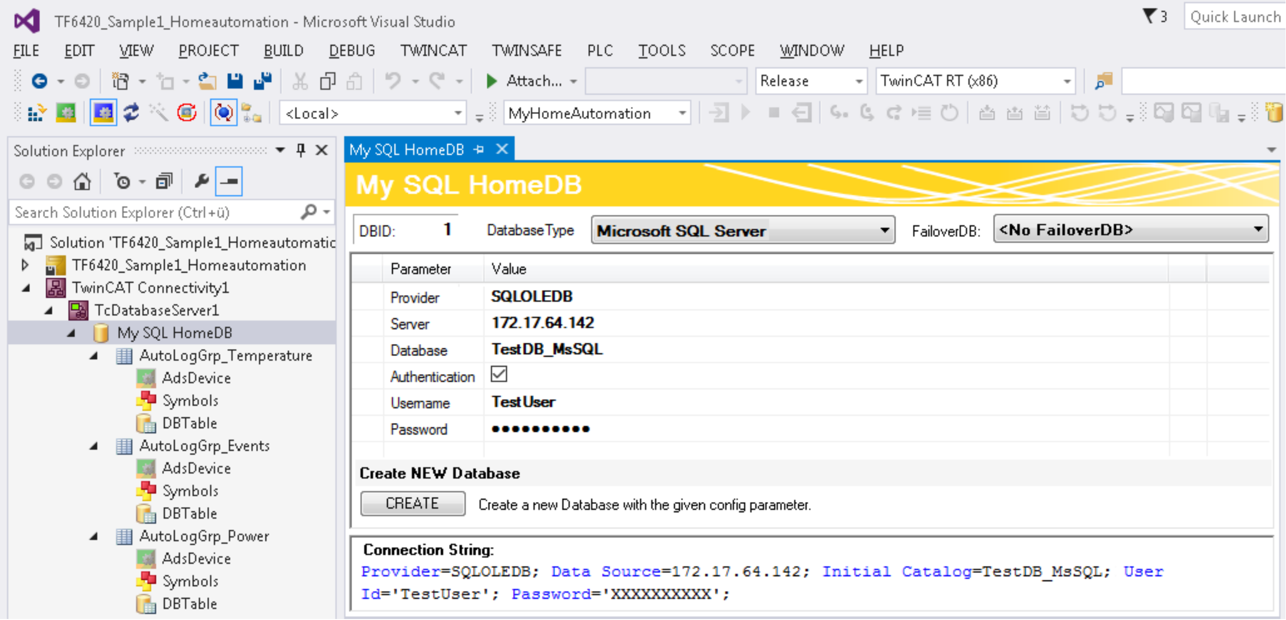
7.1.1.1 Home automation

This scenario example illustrates the configuration mode for building automation. Symbols are written to an MySQL database in three different AutoLog groups without additional programming, i.e. purely based on the configuration. Room temperatures are logged in the database at 5-minute intervals. Energy data are saved at 1-minute intervals. Events such as "lamp on", "lamp off" are stored.



Category	Configure Mode
Database used	MySQL
Compatible databases	Can be used for all supported database types
PLC function blocks used	None
PLC libraries used	Tc3_Database, Tc2_BABasic
Download	https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/9007202749926411.zip

Step 1: Once a new TwinCAT Database Server project was created, the database connection is added and configured.



My SQL HomeDB

DBID: 1 DatabaseType Microsoft SQL Server FailoverDB: <No FailoverDB>

Parameter	Value
Provider	SQLOLEDB
Server	172.17.64.142
Database	TestDB_MsSQL
Authentication	☑
Username	TestUser
Password	••••••••••

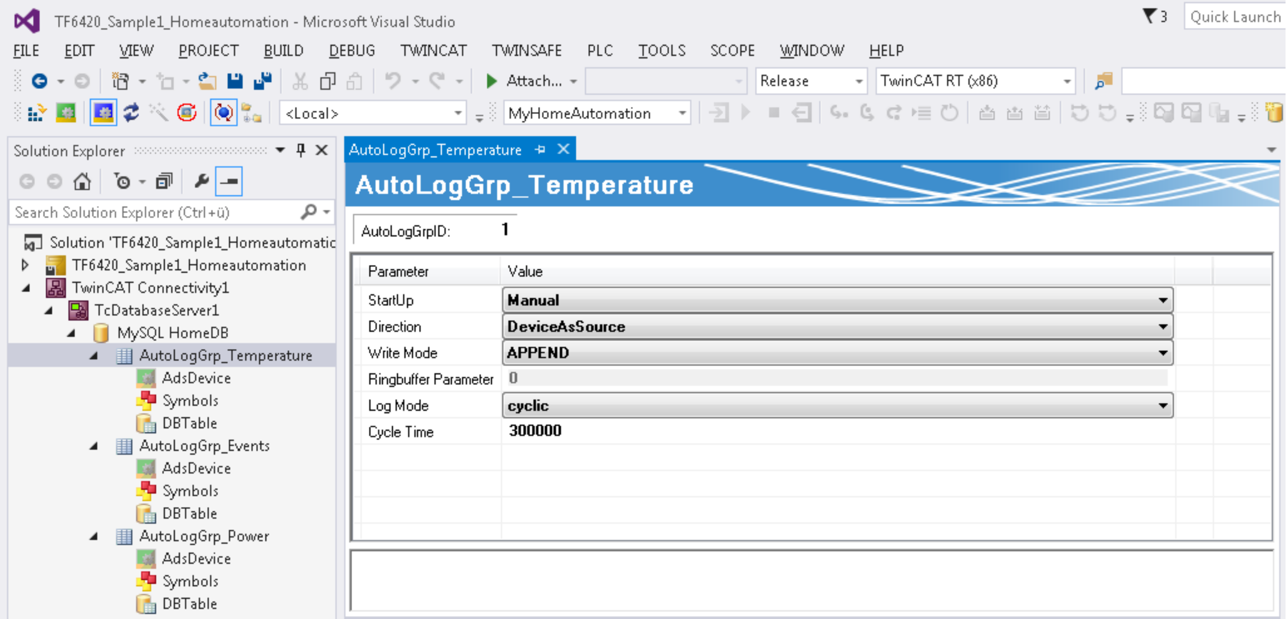
Create NEW Database

CREATE Create a new Database with the given config parameter.

Connection String:

```
Provider=SQLOLEDB; Data Source=172.17.64.142; Initial Catalog=TestDB_MsSQL; User
Id='TestUser'; Password='XXXXXXXXXX';
```

Step 2: Since logging is to take place in three different tables, three AutoLog groups are appended to the database connection. First the temperature AutoLog group is configured. The CycleTime is set to 300000 ms to match the 5-minute database logging interval.

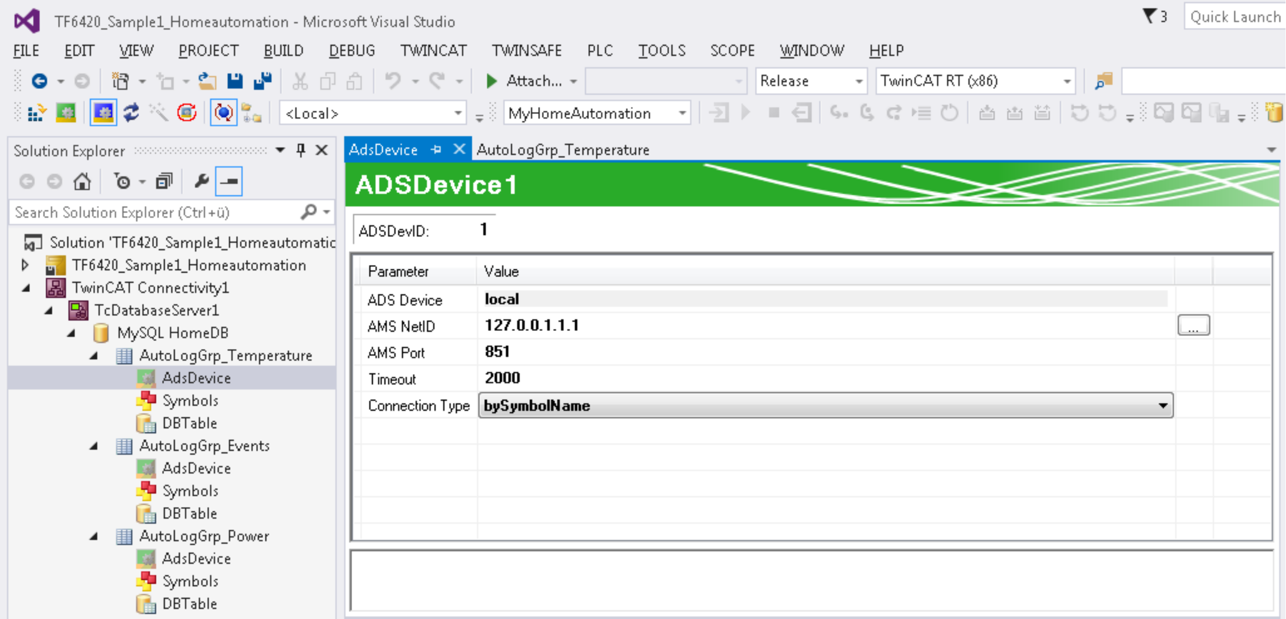


AutoLogGrp_Temperature

AutoLogGrpID: 1

Parameter	Value
StartUp	Manual
Direction	DeviceAsSource
Write Mode	APPEND
Ringbuffer Parameter	0
Log Mode	cyclic
Cycle Time	300000

Step 3: Then the ADS device from which the ADS symbols are to be read is set up.

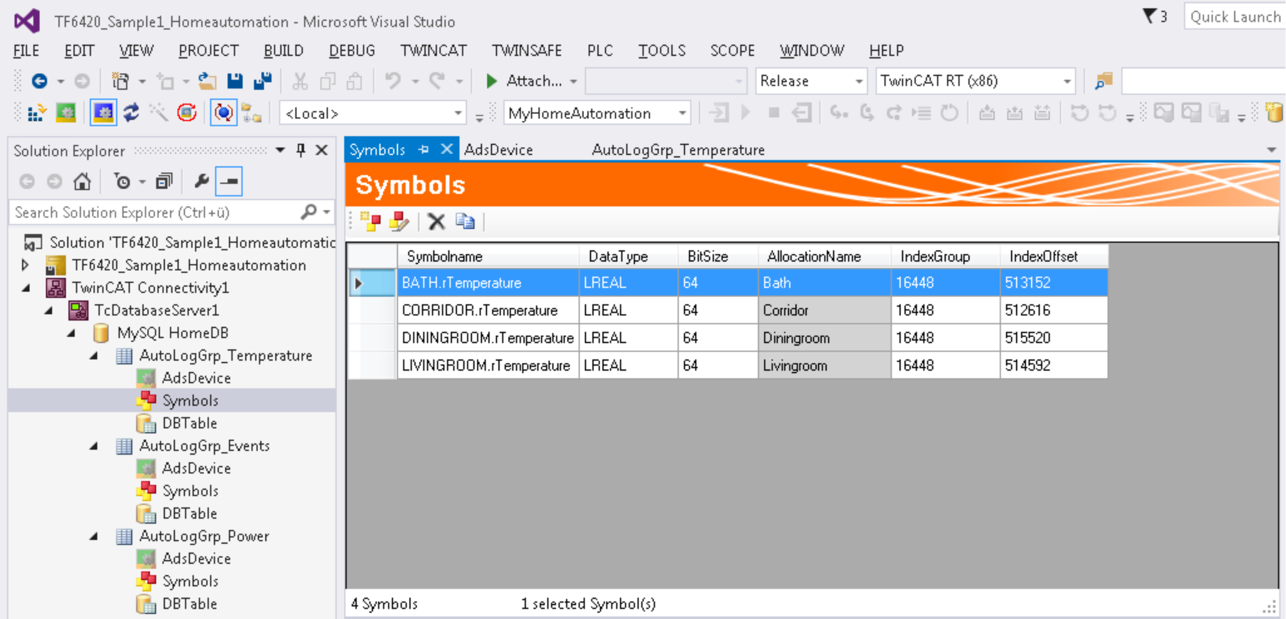


ADSDevice1

ADSDevID: 1

Parameter	Value
ADS Device	local
AMS NetID	127.0.0.1.1.1
AMS Port	851
Timeout	2000
Connection Type	bySymbolName

Step 4: The symbols are created with the target browser.

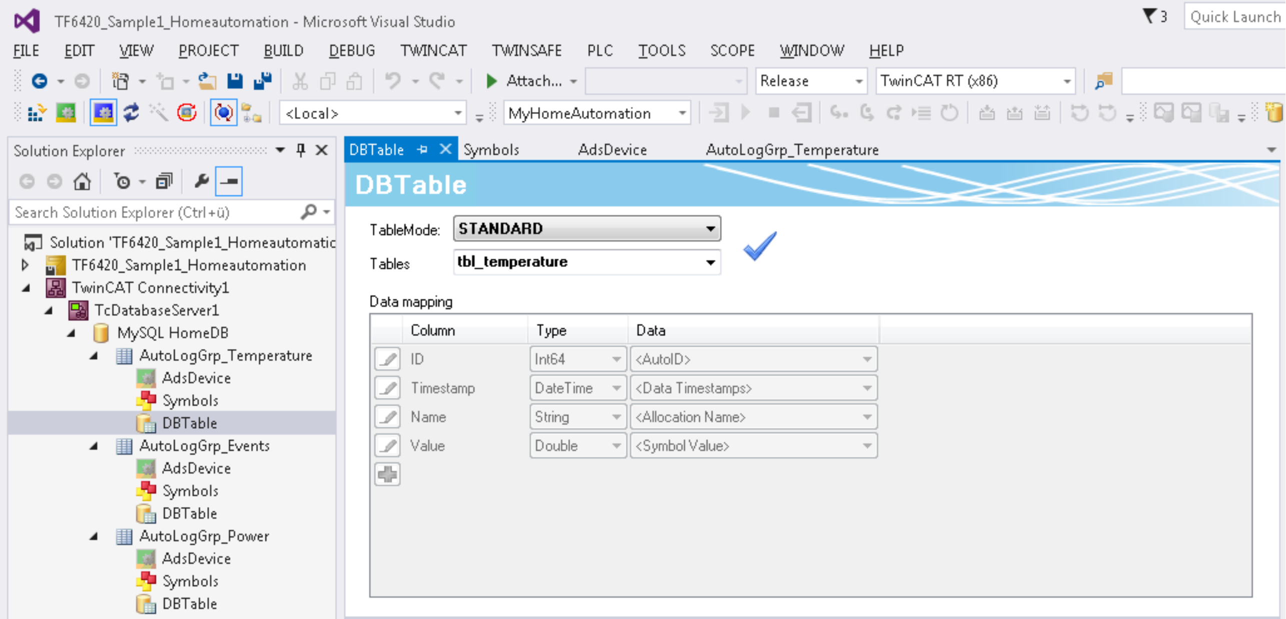


Symbols

Symbolname	DataType	BitSize	AllocationName	IndexGroup	IndexOffset
BATH.rTemperature	LREAL	64	Bath	16448	513152
CORRIDOR.rTemperature	LREAL	64	Corridor	16448	512616
DININGROOM.rTemperature	LREAL	64	Diningroom	16448	515520
LIVINGROOM.rTemperature	LREAL	64	Livingroom	16448	514592

4 Symbols 1 selected Symbol(s)

Step 5: Now you can select the table into which the temperature values are to be written. The AutoLog groups support two TableModes [▶ 41]. The standard table structure is used for the temperature values. A tick indicates that the selected table has the correct structure.



DBTable

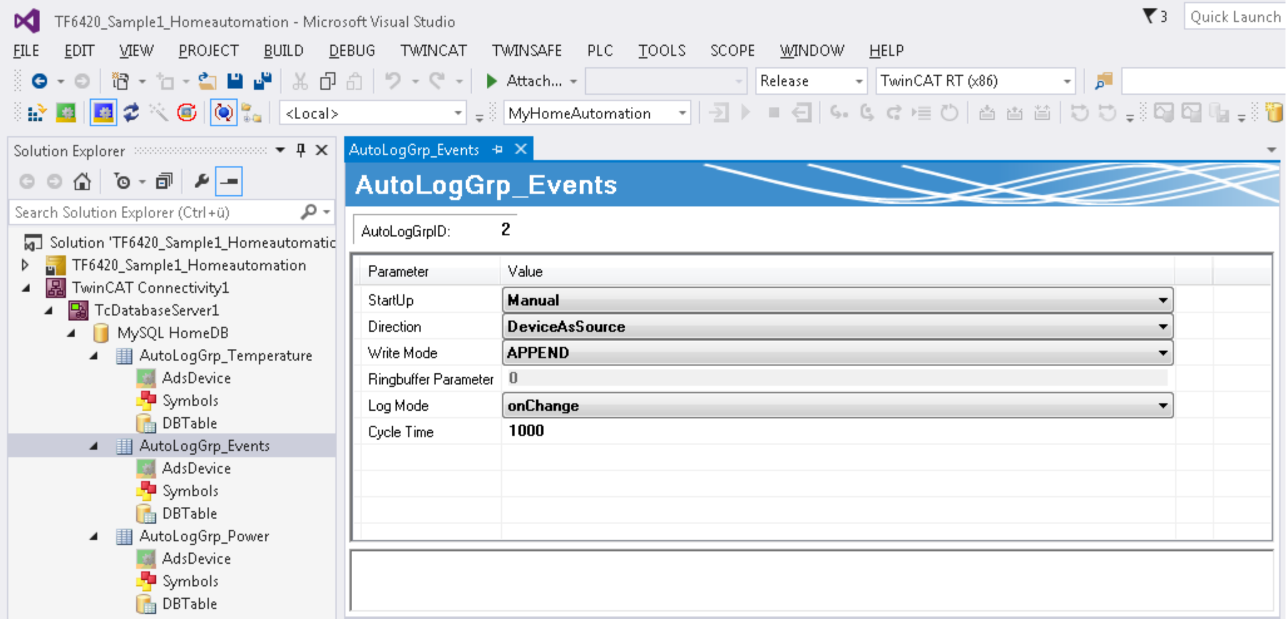
TableMode: STANDARD

Tables: tbl_temperature

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
Name	String	<Allocation Name>
Value	Double	<Symbol Value>

Step 6: Then the AutoLog group for the events is configured. The LogMode for the group is set to "onChange", the cycle time to 1000 ms. This means that the symbols are checked at 1-second intervals but a table entry is only created if a value changes.

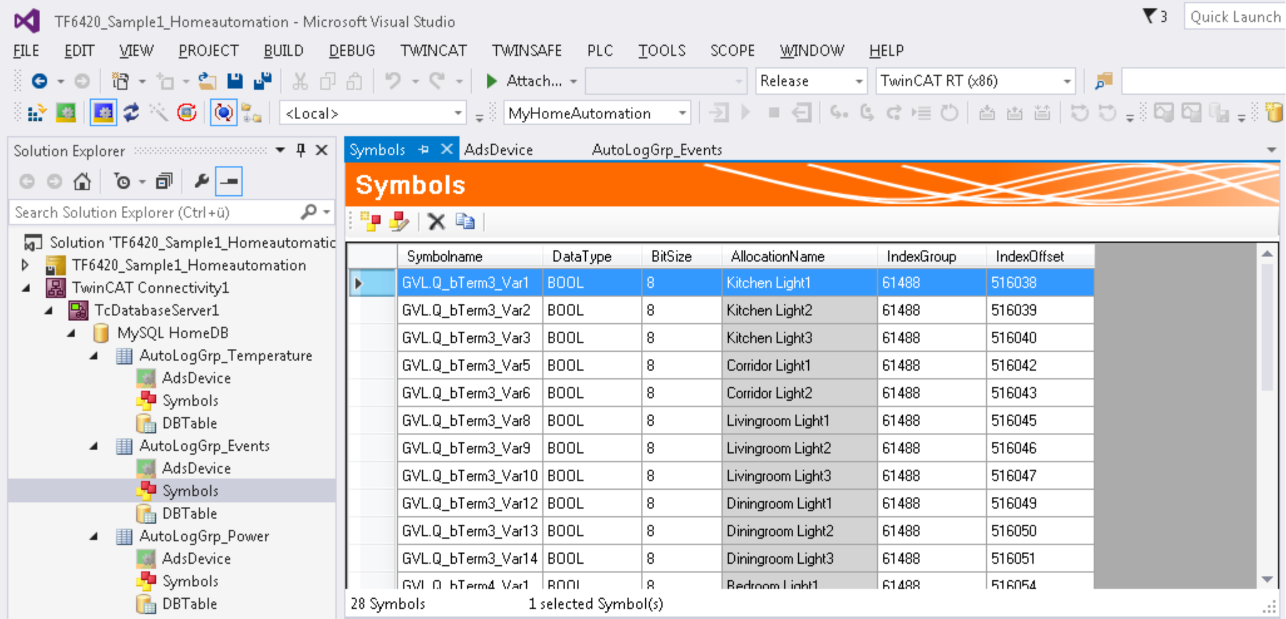


AutoLogGrp_Events

AutoLogGrpID: 2

Parameter	Value
StartUp	Manual
Direction	DeviceAsSource
Write Mode	APPEND
Ringbuffer Parameter	0
Log Mode	onChange
Cycle Time	1000

Step 7: Then the ADS device and the symbols are selected, as described in steps 3 and 4.

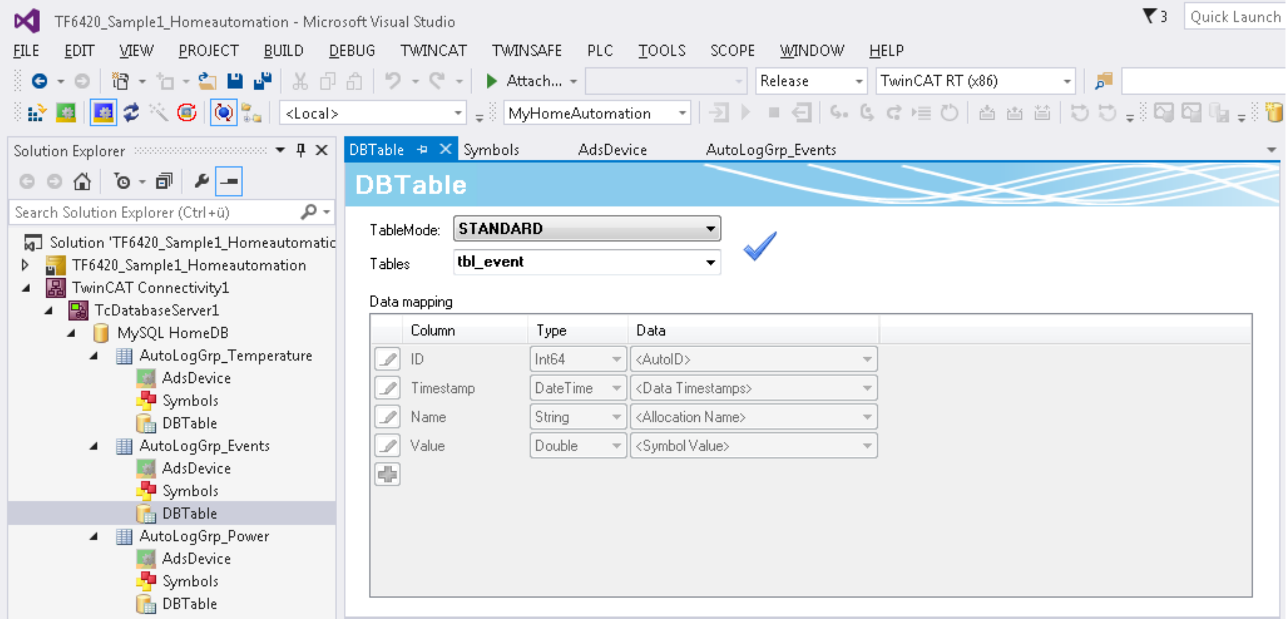

TF6420_Sample1_Homeautomation - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TWINCAT TWINSAFE PLC TOOLS SCOPE WINDOW HELP
Attach... Release TwinCAT RT (x86)
<Local> MyHomeAutomation
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TF6420_Sample1_Homeautomatic
TF6420_Sample1_Homeautomation
TwinCAT Connectivity1
TcDatabaseServer1
MySQL HomeDB
AutoLogGrp_Temperature
AdsDevice
Symbols
DBTable
AutoLogGrp_Events
AdsDevice
Symbols
DBTable
AutoLogGrp_Power
AdsDevice
Symbols
DBTable
Symbols AdsDevice AutoLogGrp_Events

Symbols

Symbolname	DataType	BitSize	AllocationName	IndexGroup	IndexOffset
GVL.Q_bTerm3_Var1	BOOL	8	Kitchen Light1	61488	516038
GVL.Q_bTerm3_Var2	BOOL	8	Kitchen Light2	61488	516039
GVL.Q_bTerm3_Var3	BOOL	8	Kitchen Light3	61488	516040
GVL.Q_bTerm3_Var5	BOOL	8	Corridor Light1	61488	516042
GVL.Q_bTerm3_Var6	BOOL	8	Corridor Light2	61488	516043
GVL.Q_bTerm3_Var8	BOOL	8	Livingroom Light1	61488	516045
GVL.Q_bTerm3_Var9	BOOL	8	Livingroom Light2	61488	516046
GVL.Q_bTerm3_Var10	BOOL	8	Livingroom Light3	61488	516047
GVL.Q_bTerm3_Var12	BOOL	8	Diningroom Light1	61488	516049
GVL.Q_bTerm3_Var13	BOOL	8	Diningroom Light2	61488	516050
GVL.Q_bTerm3_Var14	BOOL	8	Diningroom Light3	61488	516051
GVL.Q_bTerm4_Var1	BOOL	8	Bedroom Light1	61488	516054

28 Symbols 1 selected Symbol(s)

Step 8: The events are also store in a standard table structure.


TF6420_Sample1_Homeautomation - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TWINCAT TWINSAFE PLC TOOLS SCOPE WINDOW HELP
Attach... Release TwinCAT RT (x86)
<Local> MyHomeAutomation
Solution Explorer
Search Solution Explorer (Ctrl+ü)
Solution 'TF6420_Sample1_Homeautomatic
TF6420_Sample1_Homeautomation
TwinCAT Connectivity1
TcDatabaseServer1
MySQL HomeDB
AutoLogGrp_Temperature
AdsDevice
Symbols
DBTable
AutoLogGrp_Events
AdsDevice
Symbols
DBTable
AutoLogGrp_Power
AdsDevice
Symbols
DBTable
DBTable Symbols AdsDevice AutoLogGrp_Events

DBTable

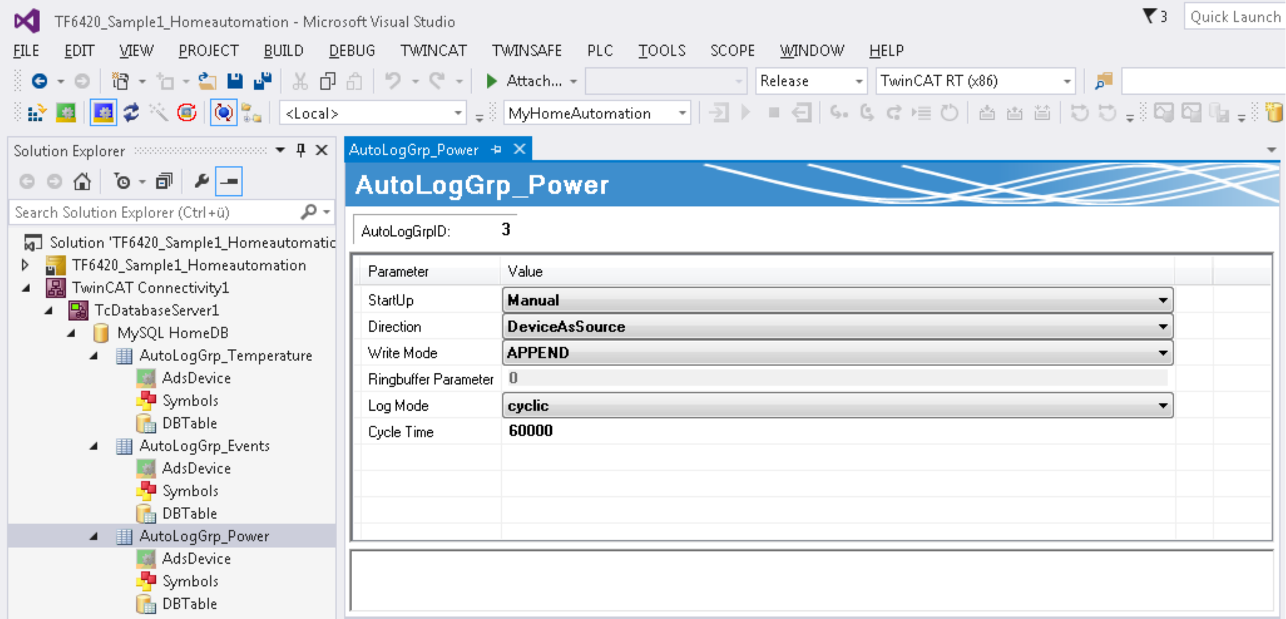
TableMode: STANDARD

Tables: tbl_event

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
Name	String	<Allocation Name>
Value	Double	<Symbol Value>

Step 9: Finally the energy data are saved at 1-minute intervals.

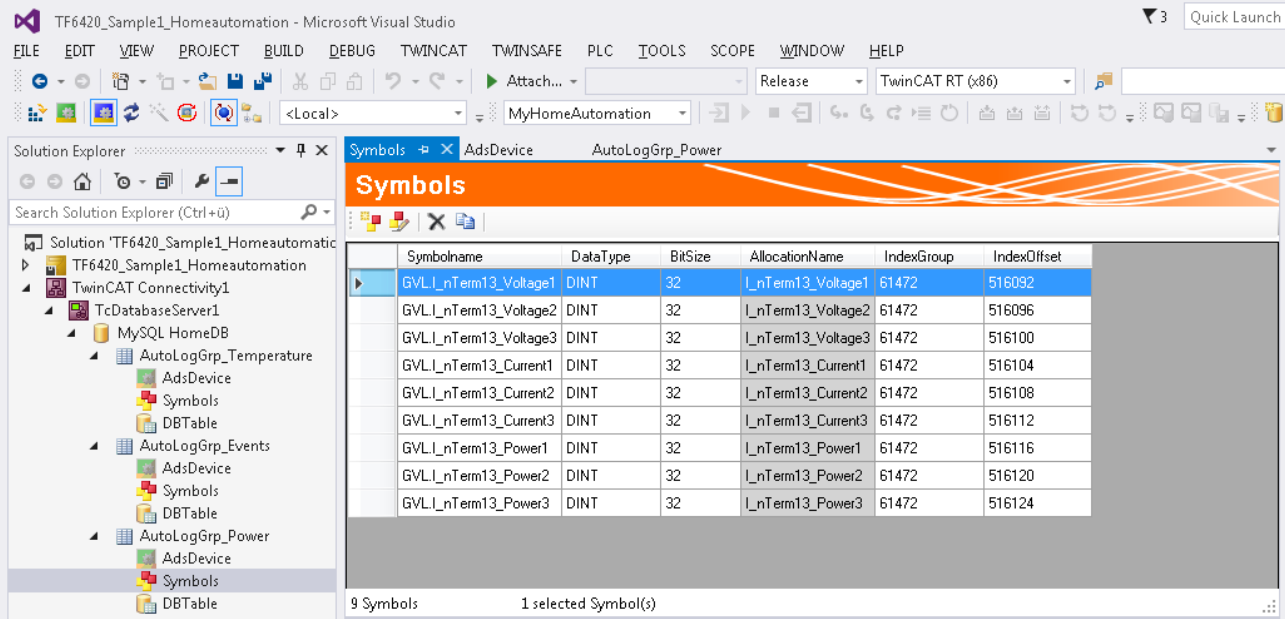


AutoLogGrp_Power

AutoLogGrpID: 3

Parameter	Value
StartUp	Manual
Direction	DeviceAsSource
Write Mode	APPEND
Ringbuffer Parameter	0
Log Mode	cyclic
Cycle Time	60000

Step 10: For the energy data the ADS device and the symbols are also selected.

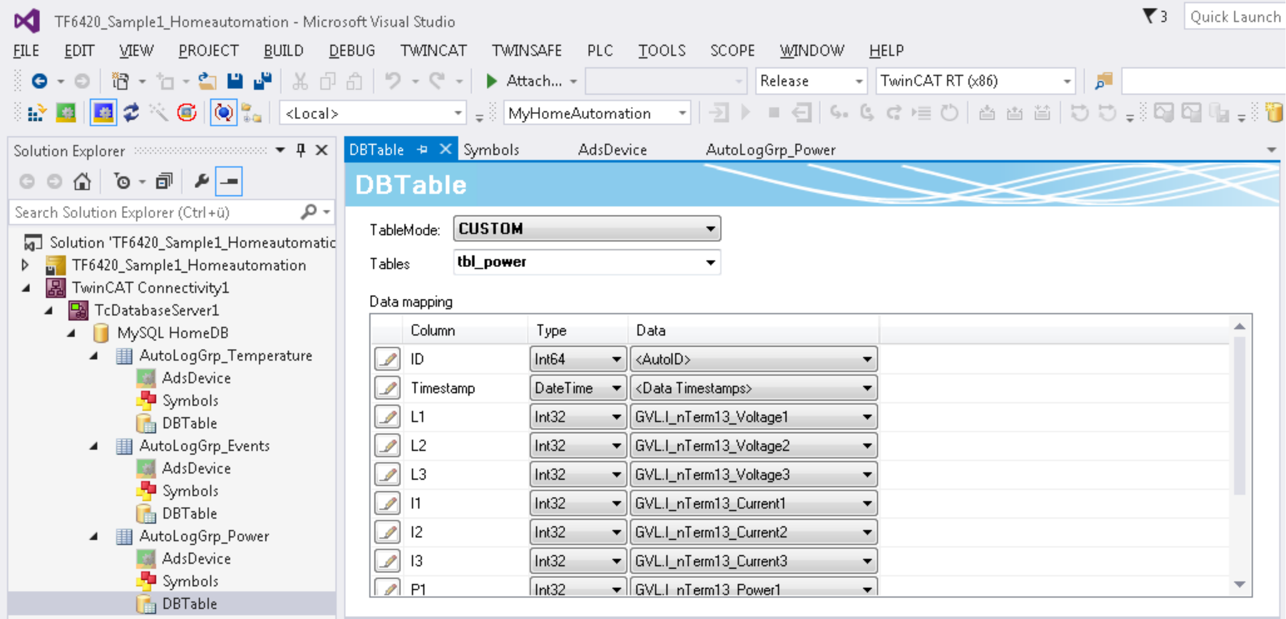


Symbols

SymbolName	DataType	BitSize	AllocationName	IndexGroup	IndexOffset
GVL.I_nTerm13_Voltage1	DINT	32	I_nTerm13_Voltage1	61472	516092
GVL.I_nTerm13_Voltage2	DINT	32	I_nTerm13_Voltage2	61472	516096
GVL.I_nTerm13_Voltage3	DINT	32	I_nTerm13_Voltage3	61472	516100
GVL.I_nTerm13_Current1	DINT	32	I_nTerm13_Current1	61472	516104
GVL.I_nTerm13_Current2	DINT	32	I_nTerm13_Current2	61472	516108
GVL.I_nTerm13_Current3	DINT	32	I_nTerm13_Current3	61472	516112
GVL.I_nTerm13_Power1	DINT	32	I_nTerm13_Power1	61472	516116
GVL.I_nTerm13_Power2	DINT	32	I_nTerm13_Power2	61472	516120
GVL.I_nTerm13_Power3	DINT	32	I_nTerm13_Power3	61472	516124

9 Symbols 1 selected Symbol(s)

Step 11: The energy data should be stored in a custom table structure, with the various symbols assigned to the columns.



DBTable

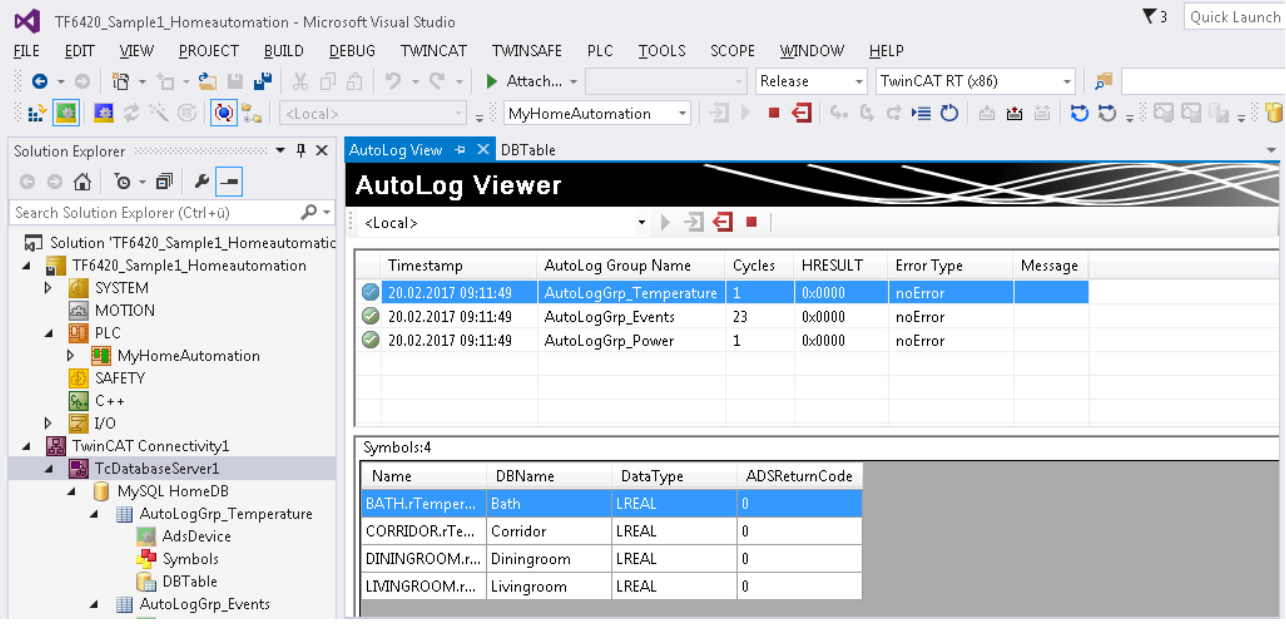
TableMode: CUSTOM

Tables: tbl_power

Data mapping

Column	Type	Data
ID	Int64	<AutoID>
Timestamp	DateTime	<Data Timestamps>
L1	Int32	GVL.I_nTerm13_Voltage1
L2	Int32	GVL.I_nTerm13_Voltage2
L3	Int32	GVL.I_nTerm13_Voltage3
I1	Int32	GVL.I_nTerm13_Current1
I2	Int32	GVL.I_nTerm13_Current2
I3	Int32	GVL.I_nTerm13_Current3
P1	Int32	GVL.I_nTerm13_Power1

Step 12: Logging can now be started and monitored with the AutoLog Viewer [▶ 52].



AutoLog Viewer

Timestamp	AutoLog Group Name	Cycles	HRESULT	Error Type	Message
20.02.2017 09:11:49	AutoLogGrp_Temperature	1	0x0000	noError	
20.02.2017 09:11:49	AutoLogGrp_Events	23	0x0000	noError	
20.02.2017 09:11:49	AutoLogGrp_Power	1	0x0000	noError	

Symbols:4

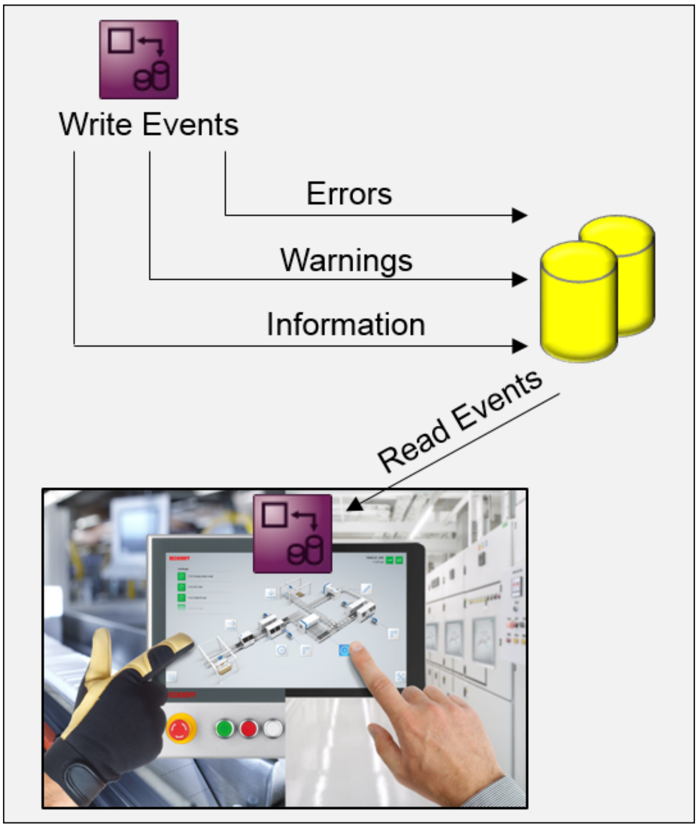
Name	DBName	DataType	ADSReturnCode
BATH.rTemper...	Bath	LREAL	0
CORRIDOR.rTe...	Corridor	LREAL	0
DININGROOM.r...	Diningroom	LREAL	0
LIVINGROOM.r...	Livingroom	LREAL	0

Documents about this

🗎 TF6420_Sample1_Homeautomation.zip (Resources/zip/9007202749926411.zip)

7.1.1.2 Message logger

This scenario example illustrates the PLC Expert Mode for a Message Logger in the PLC. In the sample program the function blocks of the TwinCAT Database Server are used to create a function block, which provides various methods for generating and reading messages. The database in which the messages are stored is created from the PLC. A sample application of the created function block is implemented in the MAIN program. A new database file is created every 7 days. Three different messages can be sent. In addition it is possible to call up the last message or all messages from a particular interval.



Category	PLC Expert Mode
Database used	MS Compact [▸ 122]
Compatible databases	Can be used with minor amendments for all supported database types
PLC function blocks used	FB_PLCDBCreateEvt [▸ 161], FB_PLCDBCmdEvt [▸ 173], FB_PLCDBWriteEvt [▸ 168], FB_PLCDBReadEvt [▸ 164]
PLC libraries used	Tc3_Database, Tc3_Eventlogger
Download	https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/9007202749928971.zip

FB_ErrorLogger

CreateErrorLogDB (method)

The CreateErrorLogDB method creates an MS Compact database file and the table in which the messages are stored.

```
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 (method)

The AddErrorEntry method can be used to write different messages into the database.

```
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 (method)

The method ReadLastError can be used to read the latest (last) entry from the database.

```
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 (method)

The method GetErrorTimerange can be used to read all messages from a particular interval.

```
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 method)

The I_Message message interface is evaluated by the TwinCAT EventLogger in the private _SetResultInfo method.

```
METHOD _SetResultInfo : BOOL
VAR_INPUT
    nLangId : INT := 1033;
END_VAR
```

```
_SetResultInfo := FALSE;

CASE nState_SetResInfo OF
```

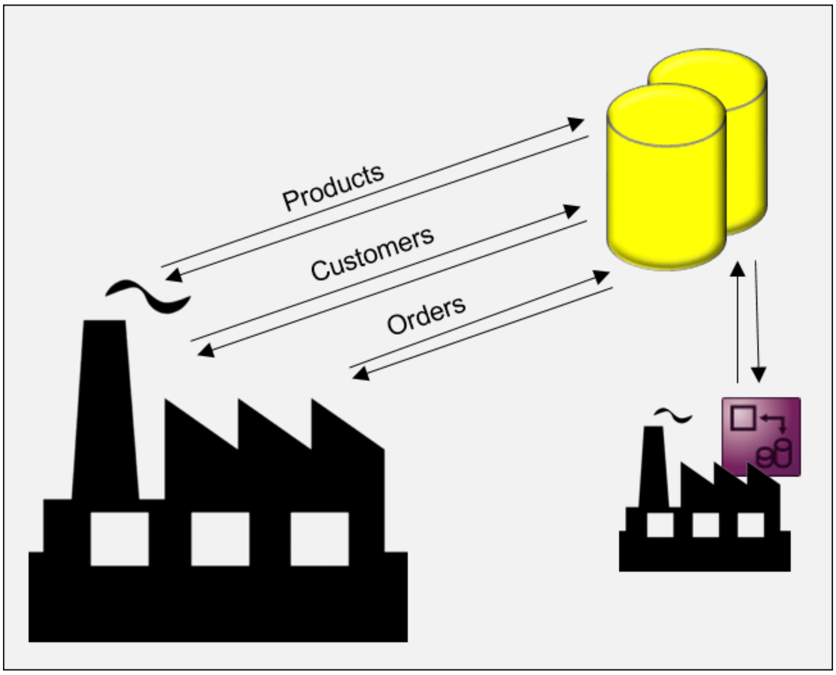
```
    0:
        IF ipResultEvt.RequestEventText(nLangId, EventText, SIZEOF(EventText)) THEN
            nState_SetResInfo := 1;
        END_IF
    1:
        IF ipResultEvt.RequestEventClassName(nLangId, EventClassName, SIZEOF(EventClassName)) THEN
            EventSourcePath := ipResultEvt.ipSourceInfo.sName;
            EventId := ipResultEvt.nEventId;

            bError := (ipResultEvt.eSeverity = TcEventSeverity.Error) OR
                     (ipResultEvt.eSeverity = TcEventSeverity.Critical);

            nState_SetResInfo:=0;
            _SetResultInfo := TRUE;
        END_IF
END_CASE
```

Documents about this

TF6420_Sample2_ErrorLogger.zip (Resources/zip/9007202749928971.zip)

7.1.1.3 Production register

This scenario example illustrates the use of the SQL Expert Mode for handling stored procedures. A connection to the database established from the PLC. A stored procedure is used to read product positions from several tables. A visualization is used for the operation.



Category	SQL Expert mode
Database used	MS SQL [▶ 120]
Compatible databases	MS SQL, MySQL, Oracle
PLC function blocks used	FB_SQLDatabaseEvt [▶ 181], FB_SQLStoredProcedureEvt [▶ 191], FB_SQLResultEvt [▶ 188]
PLC libraries used	Tc3_Database, Tc3_Eventlogger
Download	https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/18014402004672523.zip

In the MAIN program a so-called state machine is implemented for processing, through which the different SQL function blocks are controlled. Since the methods of the function blocks no longer have an Execute flag, the user must ensure that the method is not called again in the next cycle, in order to avoid repetition of the procedure. This can easily be ensured through the state machine.

```
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
```

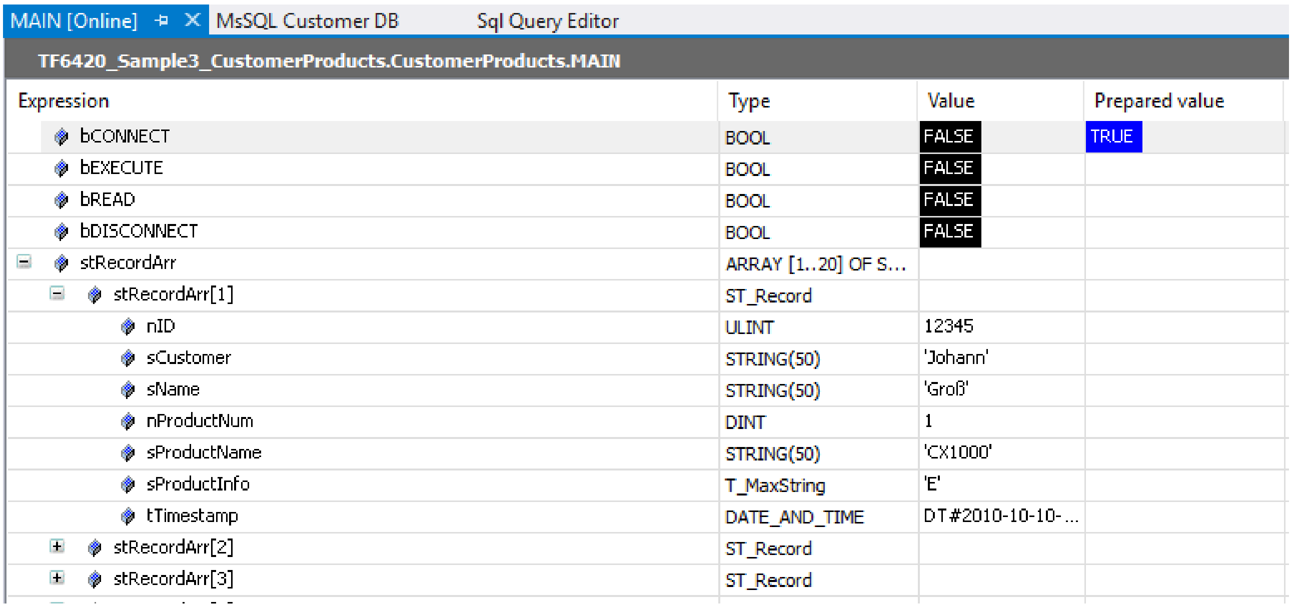
```
CASE nState OF
0:(*Idle*)
    ;
1: // Connect to database and create stored procedure instance
    CASE nState_Connect OF
        0:
            IF fbSQLDatabase.Connect(hDBID:= nDBID) THEN
                ipResultEvt := fbSQLDatabase.ipTcResult;
                bConn := NOT fbSQLDatabase.bError;
                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
```

The individual process steps can be reproduced in the individual PLC states. Boolean flags are available to facilitate handling.

1. bConnect: Connection with the database is established
2. bExecute: The stored procedure is executed, and results are loaded into the cache
3. bRead: The results are transferred to the PLC
4. bDisconnect: The connection is closed

If these steps are executed consecutively, the array stRecordArr is filled with values from the database:



TF6420_Sample3_CustomerProducts.CustomerProducts.MAIN

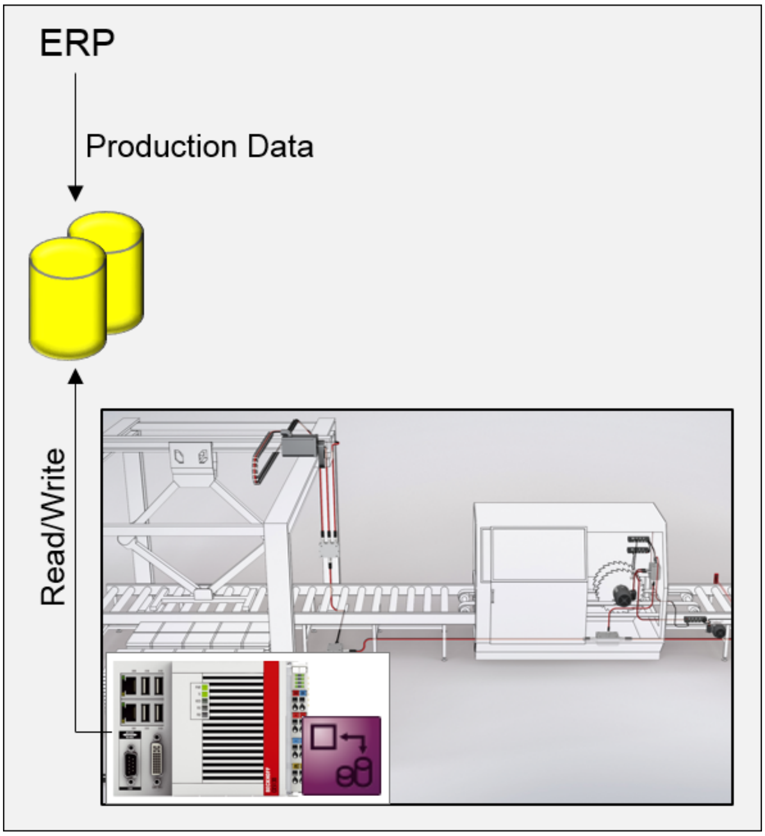
Expression	Type	Value	Prepared value
bCONNECT	BOOL	FALSE	TRUE
bEXECUTE	BOOL	FALSE	
bREAD	BOOL	FALSE	
bDISCONNECT	BOOL	FALSE	
⊟ stRecordArr	ARRAY [1..20] OF S...		
⊟ stRecordArr[1]	ST_Record		
nID	ULINT	12345	
sCustomer	STRING(50)	'Johann'	
sName	STRING(50)	'Groß'	
nProductNum	DINT	1	
sProductName	STRING(50)	'CX1000'	
sProductInfo	T_MaxString	'E'	
tTimestamp	DATE_AND_TIME	DT#2010-10-10-...	
⊞ stRecordArr[2]	ST_Record		
⊞ stRecordArr[3]	ST_Record		

Documents about this

🗎 TF6420_Sample3_CustomerProducts.zip (Resources/zip/18014402004672523.zip)

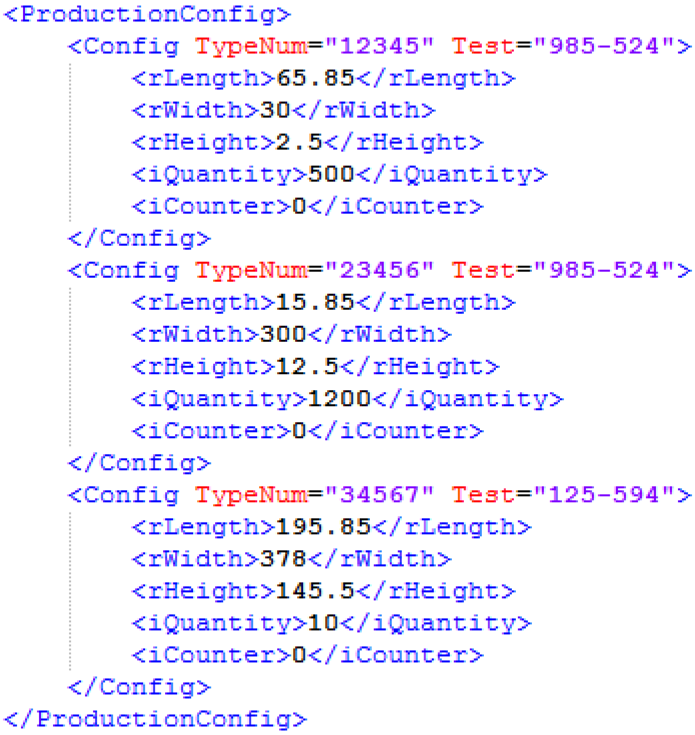
7.1.1.4 Production recipe

This scenario example illustrates how the TwinCAT Database Server handles XML files with any structure. The production recipe for building the product is read from an XML file. The corresponding test parameters are read from a different file. In addition, the test results are written into an existing XML file.



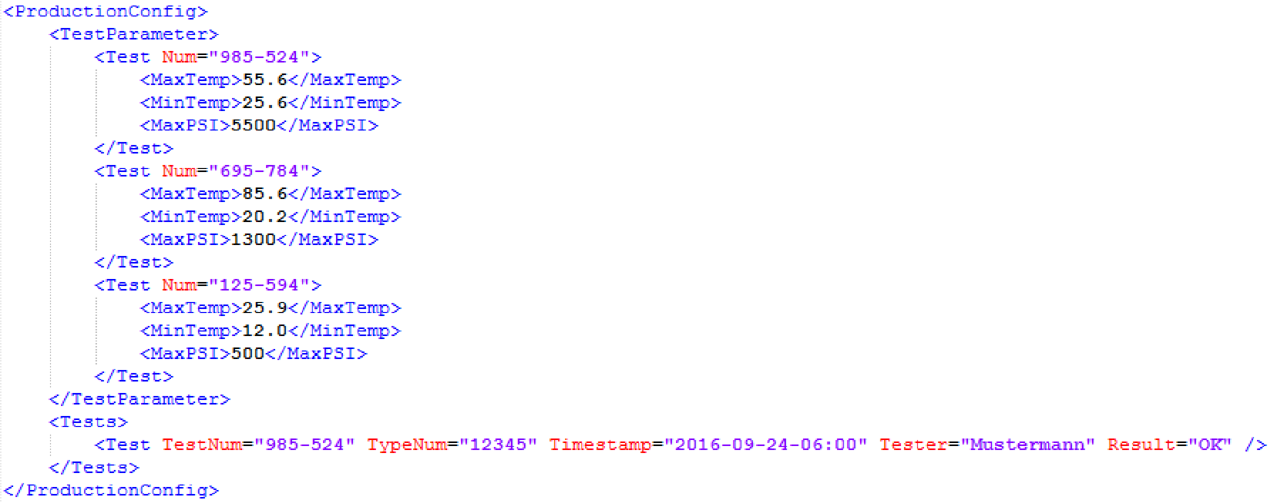
Category	SQL Expert mode
Database used	XML [▸ 128] (as free XML documents)
Compatible databases	XML
PLC function blocks used	FB_PLCDBCmdEvt [▸ 173]
PLC libraries used	Tc3_Database, Tc3_Eventlogger
Download	https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/9007202749934091.zip

Recipe 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 (method)

This method reads the production recipe for a product from an XML file. XPath queries can be used to find the required recipe.

```
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 (method)

This method reads the product-specific test parameters.

```
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 (method)

This method adds the test result to the test XML file.

```
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 method)

The I_Message message interface is evaluated by the TwinCAT EventLogger in the private _SetResultInfo method.

```
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
```

Documents about this

TF6420_Sample4_XMLProductionConfig.zip (Resources/zip/9007202749934091.zip)

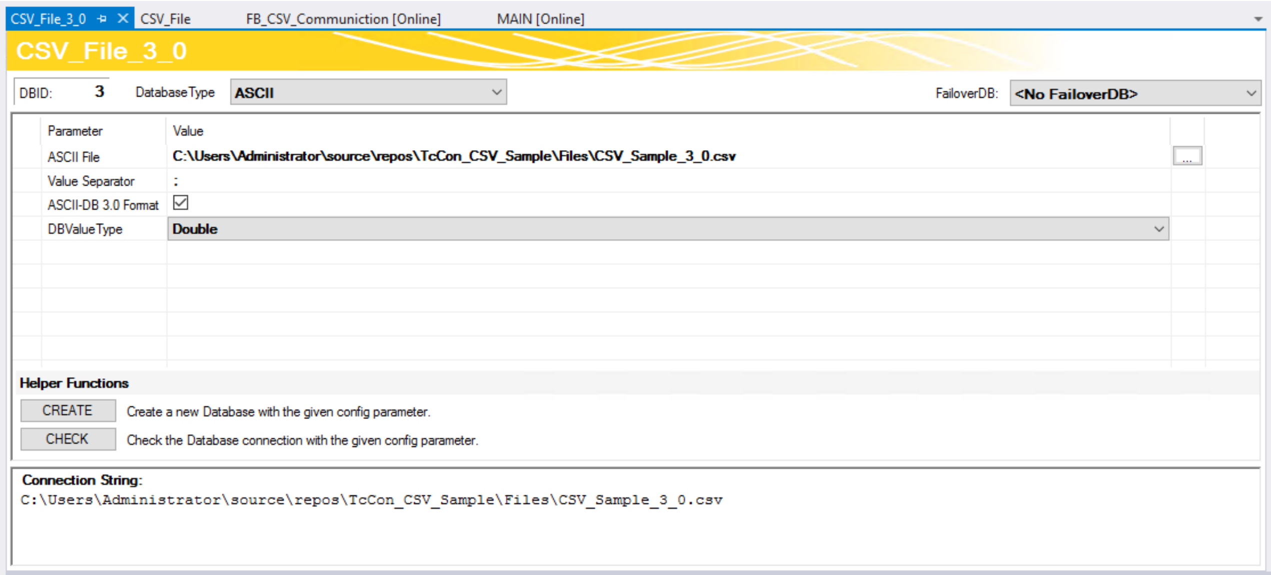
7.1.2 Best practices

The tips for using the TwinCAT Database Server illustrate the benefits of the individual function blocks and applications in terms of performance, flexibility and complexity.

7.1.2.1 Writing CSV files

The TwinCAT 3 Database Server supports the CSV file format. There are different approaches, each with advantages and disadvantages, to write content to the file or read from it. Two of these approaches are explained in more detail here.

Select the ASCII database. The .csv file format can be specified under the file path. The ASCII-DB 3.0 format flag indicates the format of the ASCII/CSV file. If the format is checked, the SAX procedure is used. With this setting, write access to the file, especially with the FB_PLCDBCmdEvt function block, is also very efficient for large files. If the format is unchecked, the DOM procedure is used, which is particularly suitable for reading a file. The data is stored in a structured form in the RAM. Therefore this method is recommended for smaller files (less than 1 MB). However, this method offers some advantages due to the structured storage. The CSV file can be used as an SQL database using a stored table structure. Use the SQL Query Editor to do this. This file can be created directly via the 'Create' button.



DBID: 3 DatabaseType ASCII FailoverDB: <No FailoverDB>

Parameter	Value
ASCII File	C:\Users\Administrator\source\repos\TcCon_CSV_Sample\Files\CSV_Sample_3_0.csv
Value Separator	;
ASCII-DB 3.0 Format	☑
DBValueType	Double

Helper Functions

CREATE Create a new Database with the given config parameter.

CHECK Check the Database connection with the given config parameter.

Connection String:
C:\Users\Administrator\source\repos\TcCon_CSV_Sample\Files\CSV_Sample_3_0.csv

Load your configuration onto your TwinCAT Database Server target system.

Table 2: ASCII format compatibility

Function block	Table structure	ASCII 3.0 format	Standard ASCII
FB_PLCDBWriteEvt.Write	standard	✓	✓
FB_PLCDBWriteEvt.WriteStruct*	variable	✗	✓
FB_PLCDBReadEvt.Read	standard	✓	✓
FB_:PLCDBReadEvt.ReadStruct*	variable	✗	✓
FB_PLCDBCmdEvt.Execute*	variable	✓	✗
FB_SQLCommandEvt	variable	✗	✓

Items marked with * are used in the following sample

High-performance writing to the CSV file

The most efficient way to write to a CSV file is based on the function block FB_PLCDBCmdEvt. To this end, the link to the CSV file must be set in ASCII-DB 3.0 format. The DBValueType is irrelevant here. A table structure does not have to be defined in advance.

Sample:

The following structure is used as an example:

```
TYPE ST_CSVDataStruct :
STRUCT
    ID: LINT;
    Timestamp: DT;
    Name: STRING(80);
    Velocity: LREAL;
    Temperature: LREAL;
END_STRUCT
END_TYPE
```

The function block is initialized as follows:

```
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
```

The individual parameters are specified in curly brackets within the command. Information about the type, byte length and name is assigned via the initialization. The name is used to recognize the parameter in the command and to replace it with the value from the PLC when it is written to the file.

The call in the PLC source code of the function block consists of a call:

```
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
```

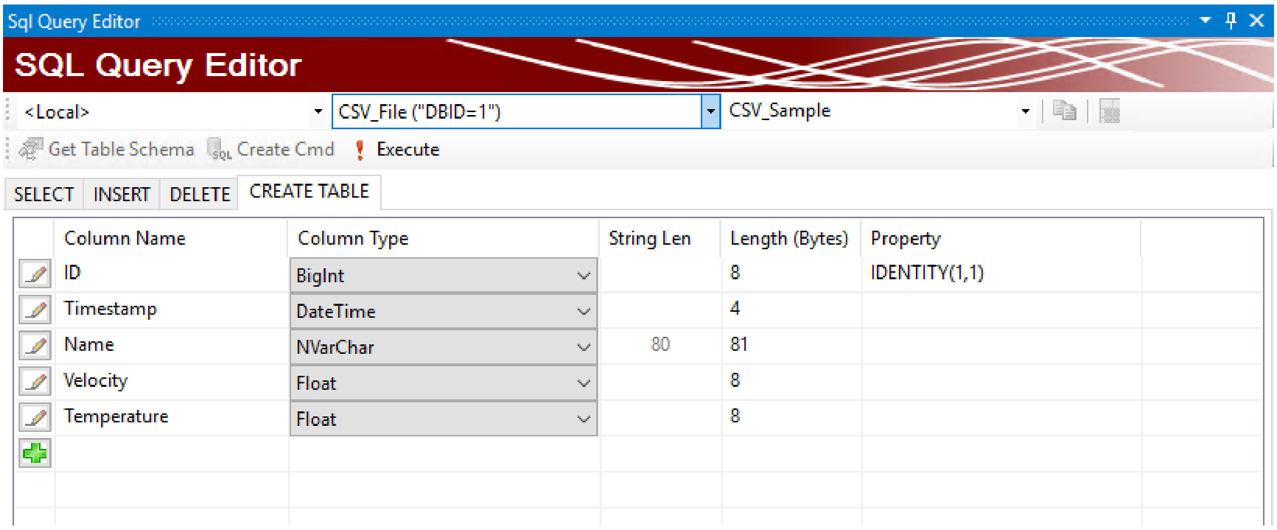
```
// Result: 16160;19-10-2018 12:27:38;Water Turbine;35.2238040741592;62.6461585412374
```

The *hDBID* depends on its configuration and can be taken from the database link. *pData* (or *cbData*) can be the address for the individual structure or for an array of its structure. This can lead to further performance improvements.

Structured writing and reading of a CSV file

Not all function blocks are possible with the ASCII format 3.0. Some functions of the TwinCAT Database Server require a preconfigured table structure. However, this cannot be stored in ASCII format 3.0. In this sample, a fixed structure is used to write and read the data with the PLCDBWriteEvt and PLCDBReadEvt function blocks in any structure.

The following structure is used as an example:



Column Name	Column Type	String Len	Length (Bytes)	Property
ID	BigInt		8	IDENTITY(1,1)
Timestamp	DateTime		4	
Name	NVarChar	80	81	
Velocity	Float		8	
Temperature	Float		8	

Export for the PLC under the 'Select' tab is also possible:

```
TYPE ST_CSVDataStruct :
STRUCT
    ID: LINT;
    Timestamp: DT;
    Name: STRING(80);
    Velocity: LREAL;
    Temperature: LREAL;
END_STRUCT
END_TYPE
```

The Write/ReadStruct methods of the respective PLC function blocks are used for any table structures:

```
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
```

The *WriteStruct(...)* method writes the *Data* structure to the database. The structures of the PLC and the CSV file are compared based on the *ColumnNames*.

The *ReadStruct(...)* method reads a certain number *(nRecordCount)* of records from the CSV file. These may be sorted based on a selected column. The size of the *ReadData* target array should be sufficient to receive all the retrieved data.

Appendix

Sample configurations for both samples, as well as the complete code of a simple sample program, can be downloaded here: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/5778536715.zip. To illustrate the process, the program generates values and repeatedly sends them to the CSV. The settings used above were stored in a separate function block, which communicates in different ways with the two CSV formats.

Documents about this

- TF6420_BestPractise_CSV.zip (Resources/zip/5778536715.zip)

7.1.2.2 Fast logging with data buffer

In order to log data in a database at millisecond intervals, the data must first be consolidated before it is transferred to the database via the TwinCAT Database Server. These data buffers can vary in size according to requirements. In the sample, 100 data samples are combined in a buffer before they are transferred with the TwinCAT Database Server. To avoid gaps during the write process, several buffers must be created in which the data samples are combined. In the sample, a total of 20 buffers are created using a 2-dimensional array.

Data sample

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
```

Each cycle fills one element of the data buffer. In the sample this happens at 10 ms intervals. Thus a buffer contains data of a period of 1 s. If a buffer is filled with 100 elements, a further array indicates that the 100 elements can now be transferred with the function block FB_PLCDBCmdEvt. To this end, the entire buffer can be transferred to the function block. Each individual element is then transferred from the TwinCAT Database Server to the database. This sample can also be implemented with other function blocks. Note that not all function blocks support arrays.

Extract from the function block 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
….
```

Appendix:

In this best practice example, a function generator block is used to generate various signals that can be logged in a database. The syntax of the INSERT command is generally valid, but has been specifically tested with an MS SQL database. The ZIP file attached below contains the complete program code in Tnzip format.

Documents about this

🗎 TF6420_BestPractise_Buffer.zip (Resources/zip/6263666699.zip)

7.1.2.3 NoSQL

This document describes the handling of NoSQL databases.

Database used: MongoDB
Database type used: DocumentDB

Data writing

Database types of type *DocumentDB* can store JSON documents with any structure. Therefore it is possible to map any structure of the PLC in *DocumentDB*s. This document can be created automatically using the FB_JSONDataType or assembled using the string blocks. Make sure that the document variable is large enough. If you want to write several documents at the same time, you can transfer them in a JSON array.

The QueryOptions [▶ 223] are defined in preparation. The collection concerned and the query type are specified for this purpose. Each query type has its own structure. The structure T_QueryOptionDocumentDB_Insert [▶ 224] is used for writing documents.

```
VAR
    fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
    InsertQueryOptions: T_QueryOptionDocumentDB_Insert;
    sDocument : STRING(1000);
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);
```

The function block FB_NoSQLQueryEvt [▶ 196] is used for writing the document into the database. The Execute() [▶ 197] method writes the transferred documents to the database. This execution is asynchronous to the PLC and can take several cycles. The Boolean return value indicates when the function block has completed its process:

```
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
```

The databases recognize the data type with which the individual variables are stored. However, as with *MongoDB,* the data type can be specified explicitly. If a timestamp is to be saved explicitly as a data type, it must be defined in the JSON document:

```
sDocument := '{…"myTimestamp": ISODate("2019-02-01T14:46:06.0000000"), …}';
```

The string can not only be formatted via the string formatting function blocks of the TwinCAT 3 libraries, but also via auxiliary function blocks for JSON documents, such as FB_JsonReadWriteDatatype from 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]));
```

Reading data

The data schema in the document-based database can be different for each document. In contrast, the PLC follows a fixed process image. The data may not correspond to the process image.

There are two different ways of reading data in the database: the find query and the aggregation method. Both return results from the database, although aggregation offers extended options for transforming the data into an appropriate form or for performing operations, such as calculating average values directly.

The QueryOptions [▶ 223] are defined in preparation. The collection concerned and the query type are specified for this purpose. Each query type has its own structure. The structure T_QueryOptionDocumentDB_Aggregation [▶ 223] is used for aggregating documents.

```
VAR
    fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
    AggregationQueryOptions: T_QueryOptionDocumentDB_Aggregation;
    sPipeStages: STRING(1000);
END_VAR
```

```
AggregateQueryOptions.pPipeStages := ADR(sPipeStages);
AggregateQueryOptions.cbPipeStages := SIZEOF(sPipeStages);
fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.Aggregation;
fbNoSQLQueryBuilder_DocumentDB.sCollectionName := 'myCollection;
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions := ADR(AggregationQueryOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(AggregationQueryOptions);
```

The FB_NoSQLQueryEvt [▶ 196] is used for sending the aggregation query. The ExecuteDataReturn() [▶ 198] method can be used to transfer the parameters and to place the returned data in the transferred memory reference. This execution is asynchronous to the PLC and takes several cycles. The Boolean return value indicates when the function block has completed its process:

```
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
```

The syntax of *sPipeStages* depends on the database type. It will return all records. Further options (with fictitious records) include:

Operator	Description
{$$match : {Place : “NorthEast”}}	All records which have "NorthEast" as value of the element "Place".
{$$project : { myValue : { $arrayElemAt : ["$WindPlantData.RotorSensor", 2]} } }	Returns all RotorSensor data from array element location 2 as "myValue".
{$$project : {RotorAvg : {$avg: "$WindPlantData.RotorSensor"} } }	Returns the average value of the data array "RotorSensor" as "RotorAvg".

The complete documentation of the operators is available from the respective database provider.

A reference to the returned data can now be found in the function block FB_NoSQLResultEvt [▶ 199]. These can now be read as JSON documents in a string or as a structure. The data is now read directly into an array with a suitable structure. You can use the SQL Query Editor of the Database Server to directly generate a structure that matches the record. Instead of an array, it is also possible to store an address for a single structure when retrieving only one record.

```
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
```

The TwinCAT Database Server takes into account the names of the elements in the record and the names of the variables when assigning record or structure. If these are to differ, the attribute "ElementName" can be used in the PLC:

```
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
```

In this sample, "ElementName" specifies the name of the data in the database document. The start index and the number of records can be used to determine which records are to be returned with this call. In order to avoid possible duplications, please note that these options can already be carried out with operators at the "PipeplineStages".

Data validation

If there were conflicts between the record and the structure in the PLC at FB_NoSQLResult [▶ 199], they can be read out with FB_NoSQLValidationEvt [▶ 203]. Examples of conflicts are missing or surplus records, or data type problems. The method GetIssues() [▶ 204] can be used to read all conflicts as an array of strings. Surplus data that were not found in the PLC structure can be read as an array of strings in JSON format via GetRemainingData() [▶ 204]. If necessary, these can then be read out separately into the correct structure or interpreted via the TwinCAT JSON library.

```
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
```

7.2 Tc2_Database

All sample applications for the TwinCAT Database Server were consolidated in a solution. The solution can be downloaded here from a central location: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip

In addition to the tszip file for the TwinCAT 3 solution, the zip file contains all the required file-based databases. If the folder "Samples" from the zip file is located in the default installation folder: *C:\TwinCAT\Functions\TF6420-Database-Server\Win32*, the paths in the Database Server configuration do not have to be edited further. The samples with non-file-based databases, such as MS SQL, have to be individual adapted with the configurator.

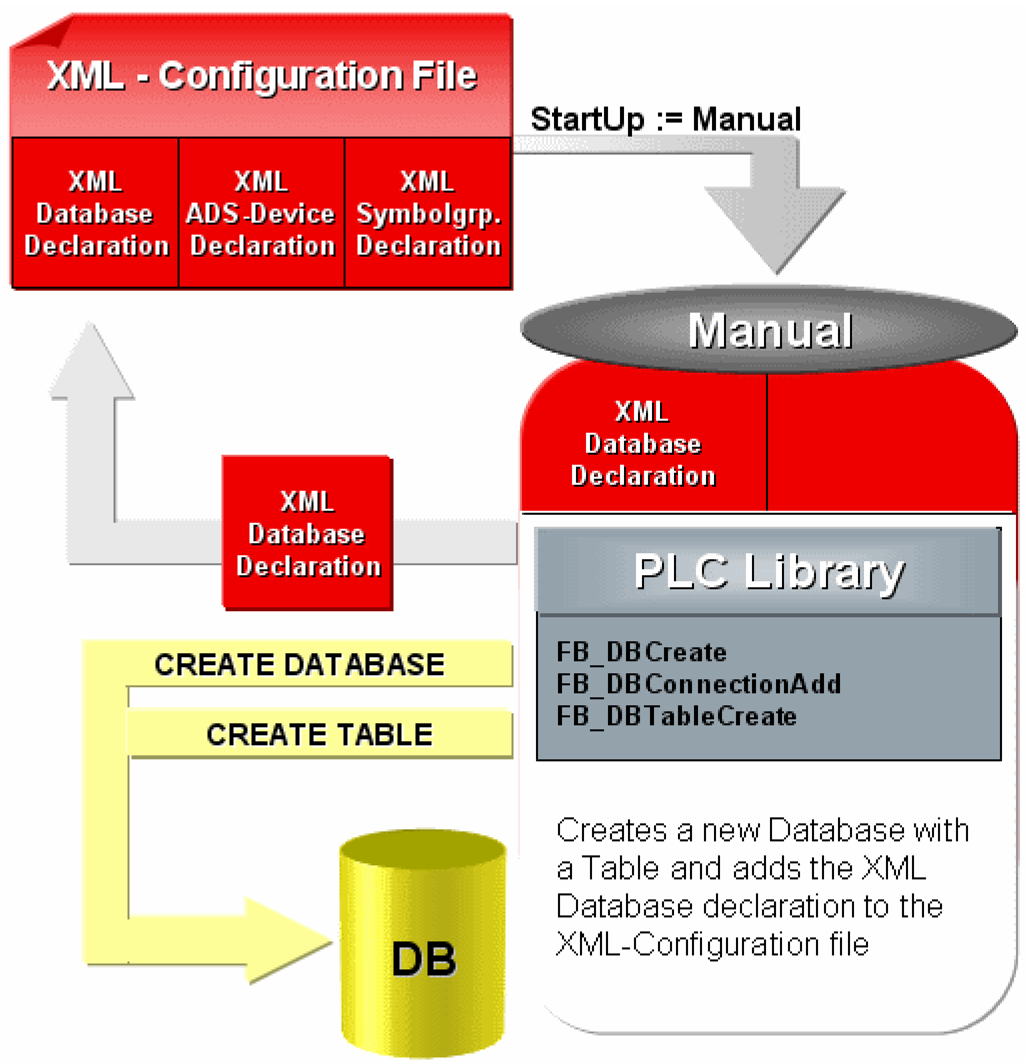
The individual samples are documented in detail on separate pages:

SP project name	Description
Create_DB_Sample [▸ 343]	Creating a database connection and a table from the PLC
Cyclic_RdWrt_Sample [▸ 346]	Cyclic logging/writing to/from a database
Write_DB_Sample [▸ 348]	Writing of variables into a database with a simple PLC function block without SQL command
SQL_InsertSelect_Sample [▸ 352]	Sample with function block FB_DBRecordInsert/ FB_DBRecordArraySelect
StoredProcedures_Sample [▸ 354]	Stored procedures with FB_DBStoredProceduresRecordArray
XML_DB_Sample [▸ 357]	Using XML files as database
XML_XPath_Sample [▸ 362]	XML XPath sample without schema
XML_XPath_Schema_Sample [▸ 365]	XML XPath sample with XML Schema, comparable with TwinCAT XML Server "Read"

7.2.1 Creating an MS Access database

This example illustrates the creation of a database from the PLC. In addition, a table is added, and the database that has been generated is declared in the XML configuration file.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip



Database type used	MS Access
Compatible database types	MS SQL, MS Compact SQL, MS Access, XML
Function blocks used	FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate
Libraries to be integrated	Tc2_Database, Tc2_System, Tc2_Standard
Download file list	TcDBSrv_InfoSysSamples.tszip

A table with the name "myTable", which has the following structure, is added to the generated database:

Column name	Data type	Property
ID	Bigint	IDENTITY(1,1)
Timestamp	datetime	
Name	Ntext	
Value	Float	

This table structure is generated with the following array:

```
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)];
```

Variable Declaration

```
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
```

PLC program

```
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
```

In order to use this sample, you only need to transfer the NetID of the ADS device (on which the TwinCAT Database Server is installed) to the sNetID input.

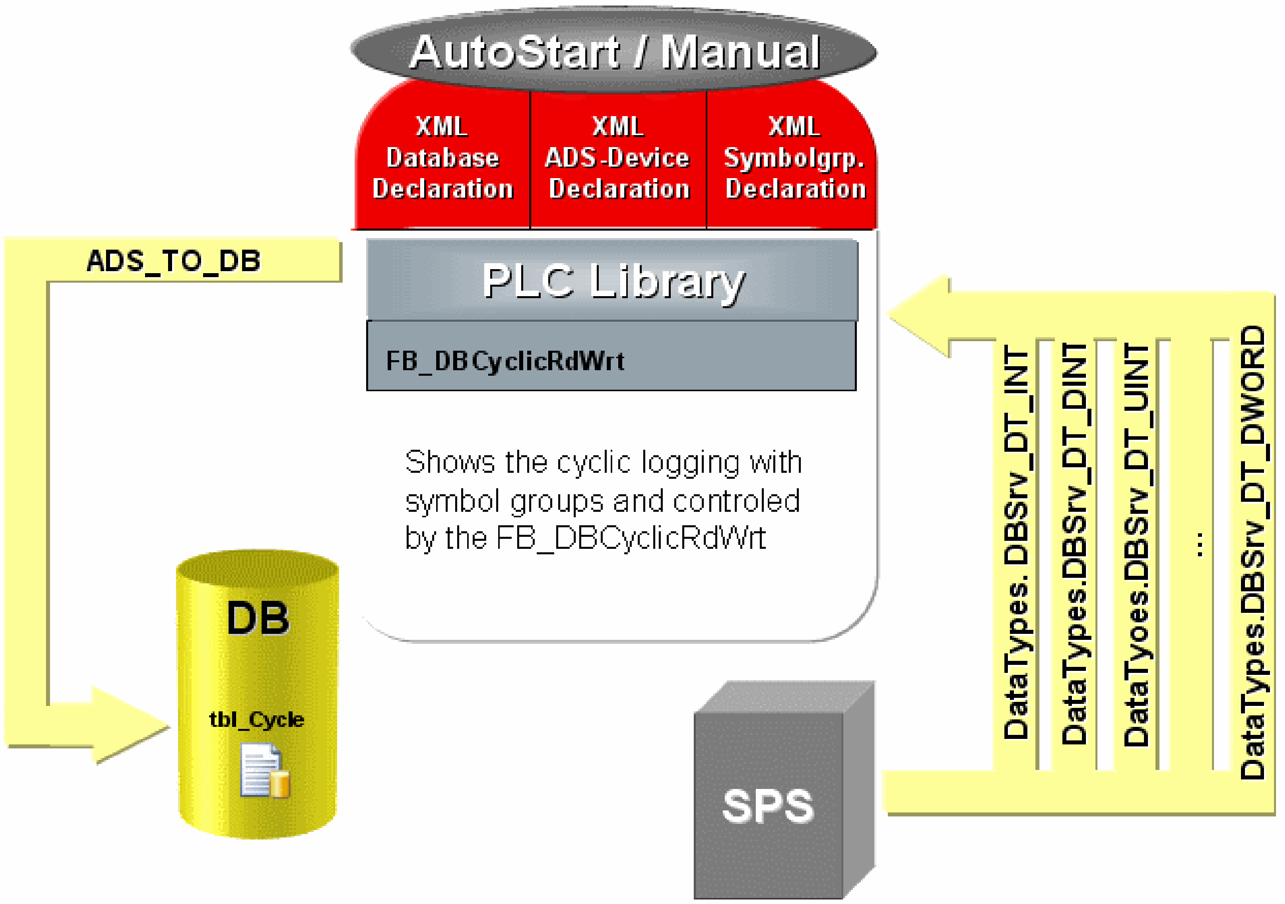
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

7.2.2 Starting / stopping, cyclic logging

This sample illustrates the starting and stopping of cyclic logging from the PLC.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip



Database type used	MS Compact SQL
Compatible database types	ASCII, MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/ Firebird, XML
Function blocks used	FB_DBCyclicRdWrt
Libraries to be integrated	Tc2_Database, Tc2_System, Tc2_Standard
Download file list	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml, TestDB_Cyclic.sdf

In this sample the cyclic log function is started or stopped by toggling the bStartStop variable.
The cyclic log process begins in response to a positive edge at the bExecute input.
A negative edge will end the process again.

Variable declaration (PRG data types)

```
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
```

ST_MyStruct structure

```
TYPE ST_MyStruct :
STRUCT
    iValue1 : INT;
    iValue2 : UINT;
    iValue3 : BOOL;
    iValue4 : REAL;
END_STRUCT
END_TYPE
```

Variable Declaration

```
PROGRAM MAIN
VAR
    fbDBCyclicRdWrt1: FB_DBCyclicRdWrt;

    bCyclic         : BOOL :=TRUE;

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

PLC program

```
DataTypes;

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

In order to use this sample, you only need to transfer the NetID of the ADS device (on which the TwinCAT Database Server is installed) to the sNetID input.
When you run the program and set the bCyclic variable to TRUE, all the variables that are declared in the symbol group of the XML configuration file are logged.

TwinCAT Database Server

All Microsoft SQL Compact databases, which are declared in the XML configuration file, must exist. They are not generated automatically.

The declared ASCII files, on the other hand, are generated automatically if they do not exist.

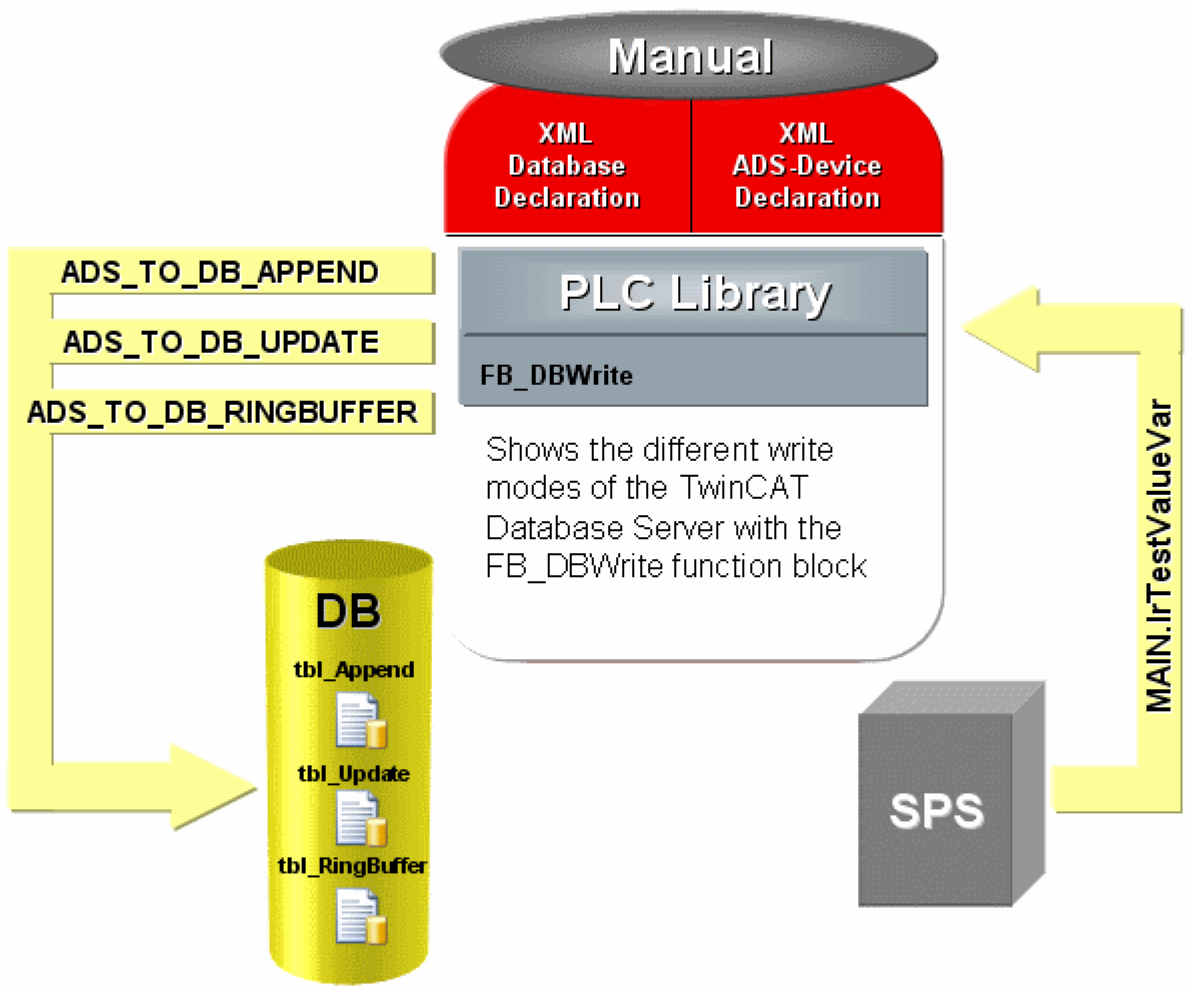
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

7.2.3 Logging of a PLC variable with FB_DBWrite

This sample illustrates logging of a PLC variables from the PLC in a database and the operating principle of the individual write modes.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip



Database type used	MS SQL
Compatible database types	ASCII, MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/ Firebird, XML
Function blocks used	FB_DBWrite
Libraries to be integrated	Tc2_Database, Tc2_System, Tc2_Standard
Download file list	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml, SQLQuery.sql

In order to be able to use this sample, you have to adapt the server name and the authentication in the XML configuration file (CurrentConfigDataBase.xml). Ensure that no "TestDB" database is present before executing the SQLQuery.sql script.

Sample configuration:

The variable "eWriteMode" can be used to set the write mode for logging.
The write operation can then be started with a positive edge at the variable "bSTART".

Table assignment:

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

Table structure used

Column name	Data type	Null permitted	Feature
ID	Bigint	no	IDENTITY(1,1)
Timestamp	datetime	no	
Name	Ntext	no	
Value	Float	no	

Variable Declaration

```
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
```

PLC program

```
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

All Microsoft SQL Compact databases, which are declared in the XML configuration file, must exist. They are not generated automatically.

The declared ASCII files, on the other hand, are generated automatically if they do not exist.

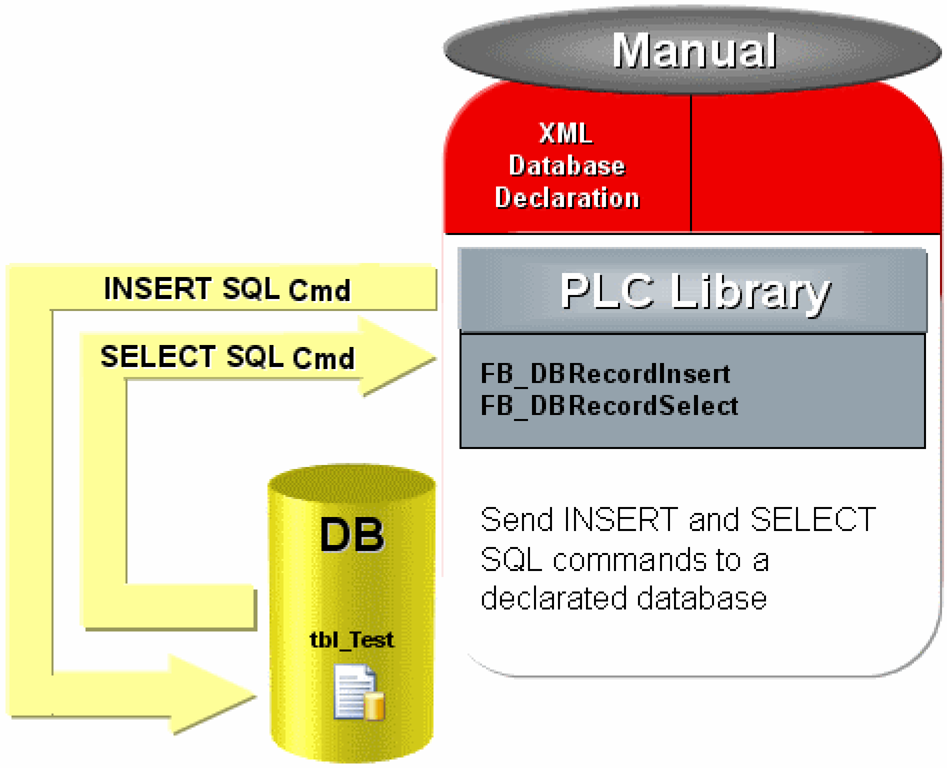
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

7.2.4 Example with the FB_DBRecordInsert and FB_DBRecordSelect function blocks

This example illustrates logging of several values in a database from the PLC with the function block FB_DBRecordInsert. In this example, several PLC variables are logged in a single record. In addition, the function block FB_DBRecordSelect can be used to read a record from this database.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip



Database type used	MS Access
Compatible database types	MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird, XML
Function blocks used	FB_DBRecordInsert, FB_DBRecordSelect
Libraries to be integrated	"**Tc2_Database**", "Tc2_System", "Tc2_Standard", "Tc2_Utilities"
Download file list	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml, TestDB_Access.mdb

The following table structure is used for writing:

Column name	Data type
Timestamp	datetime
PLC_TestValue1	float
PLC_TestValue2	float
PLC_TestValue3	float
PLC_TestValue4	String

Variable Declaration

```
(* 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
```

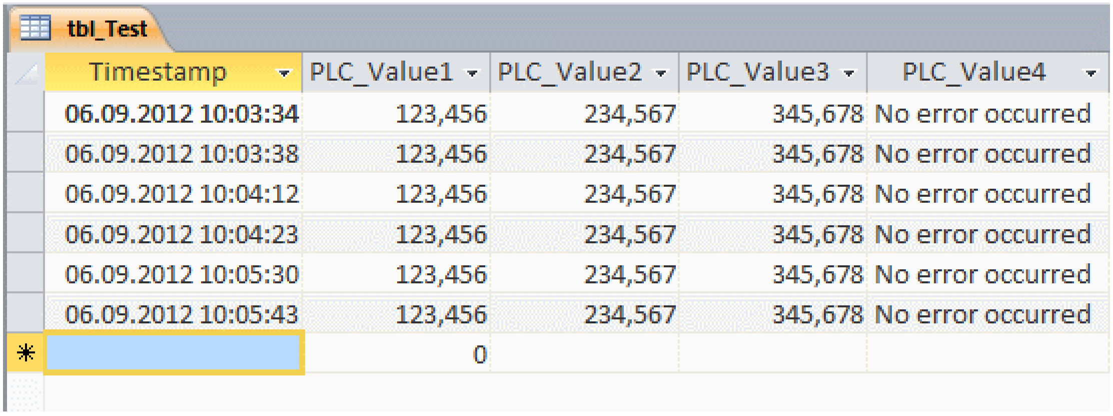
Struct ST_Record

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

PLC program

```
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
```

To use this sample, you have to declare the Access database "Sample7.mdb" in the XML configuration file. A record with the four PLC values and the timestamp is created in the database by generating a positive edge at the variable "bStartstopInsert".



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			

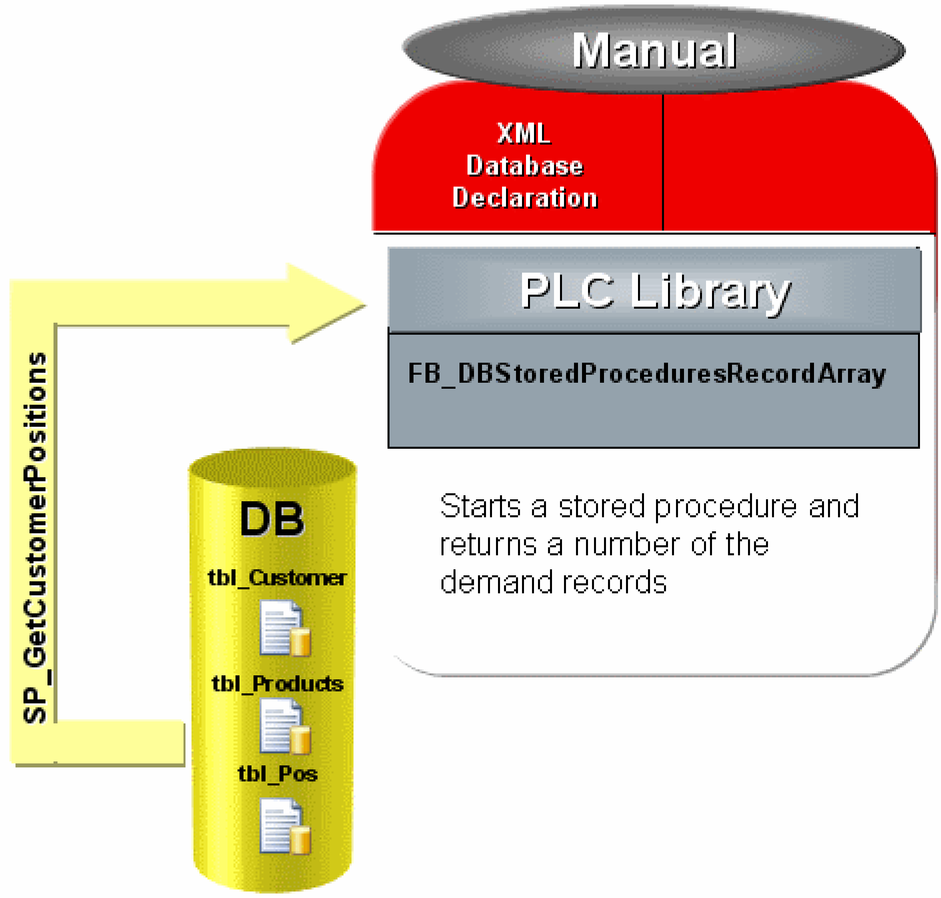
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

7.2.5 Stored procedures with FB_DBStoredProceduresRecordArray

The function block FB_DBStoredProceduresRecordArray can be used to declare parameters as INPUT, OUTPUT or INOUT and transfer them to the stored procedures. In this way complex SQL commands can be preprogrammed in the database server and then triggered by the TwinCAT Database Server. In contrast to the function block FB_DBStoredProceduresRecordReturn, this function block can be used to return several records with a single call.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip


Manual
XML Database Declaration
PLC Library
FB_DBStoredProceduresRecordArray
Starts a stored procedure and returns a number of the demand records
SP_GetCustomerPositions
DB
tbl_Customer
tbl_Products
tbl_Pos

Database type used	MS SQL (MS SQL Server 2008)
Compatible database types	MS SQL, MySQL, Oracle
Function blocks used	FB_DBStoredProceduresRecordArray
Libraries to be integrated	Tc2_Database, Tc2_System, Tc2_Base, Tc2_Utilities
Download file list	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml

The following sample illustrates the call in a simple stored procedure with an input parameter and return record. The procedure was created on a Microsoft SQL Server 2008.

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
```

Variable declaration in the PLC

```
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
```

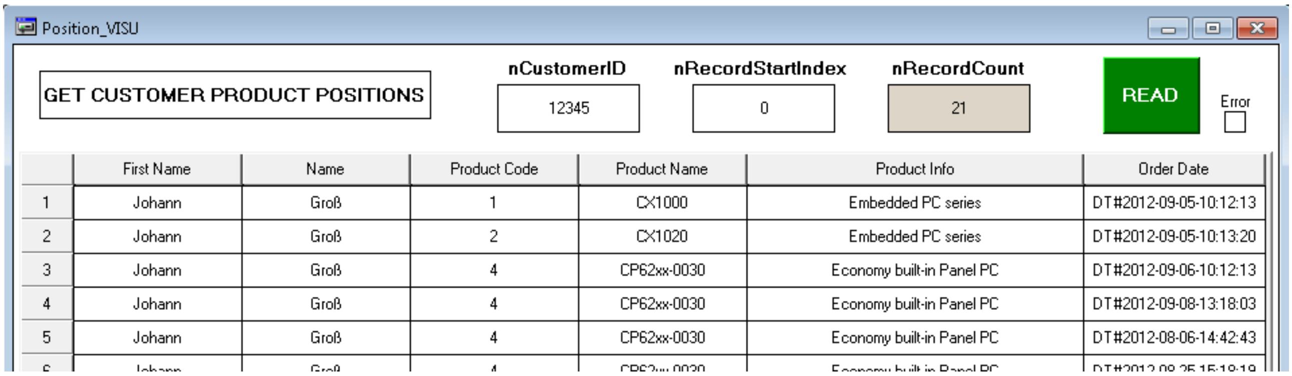
Record structure in the PLC (ST_Record)

```
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
```

PLC program

```
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
```

Visualization



GET CUSTOMER PRODUCT POSITIONS

nCustomerID: 12345
nRecordStartIndex: 0
nRecordCount: 21

	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
6	Johann	Groß	4	CP62xx-0030	Economy built-in Panel PC	DT#2012-08-25-15:19:19

Requirements

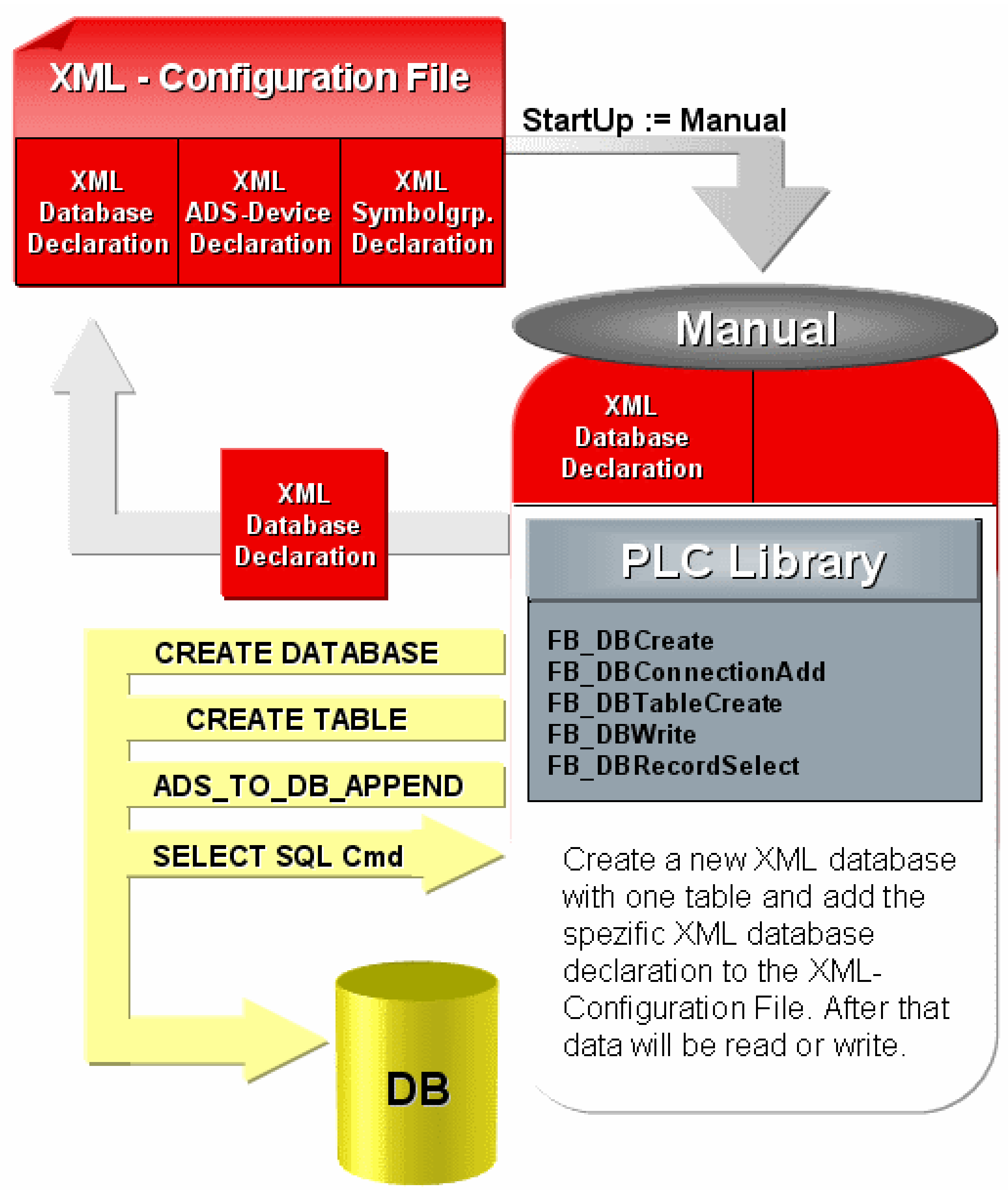
Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

7.2.6 Using XML as database

The TwinCAT Database Server provides the ability to use an XML file as a database. Apart from the "Stored Procedure" functions, the XML database type supports all known function blocks for reading and writing in a database. Even SQL commands that can be issued with the function blocks FB_DBRecordInsert or FB_DBRecordSelect are interpreted by the TwinCAT Database Server and applied to the XML file.

This sample demonstrates how an XML database is created, filled with the function block FB_DBWrite and subsequently read with an SQL SELECT command and the function block FB_DBRecordSelect.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip



Database type used	XML
Compatible database types	MS SQL, MS Compact SQL, MS Access, XML
Function blocks used	FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate, FB_DBWrite, FB_DBRecordSelect
Libraries to be integrated	"**Tc2_Database**", "Tc2_System", "Tc2_Standard", "Tc2_Utilities"
Download file list	TcDBSrv_InfoSysSamples.tszip

MAIN program

```
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
```

The process is started with a positive edge at the toggle variable bSTART.

The following files are created:

XMLTestDB.xml (XML database file)

```
<?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>
```

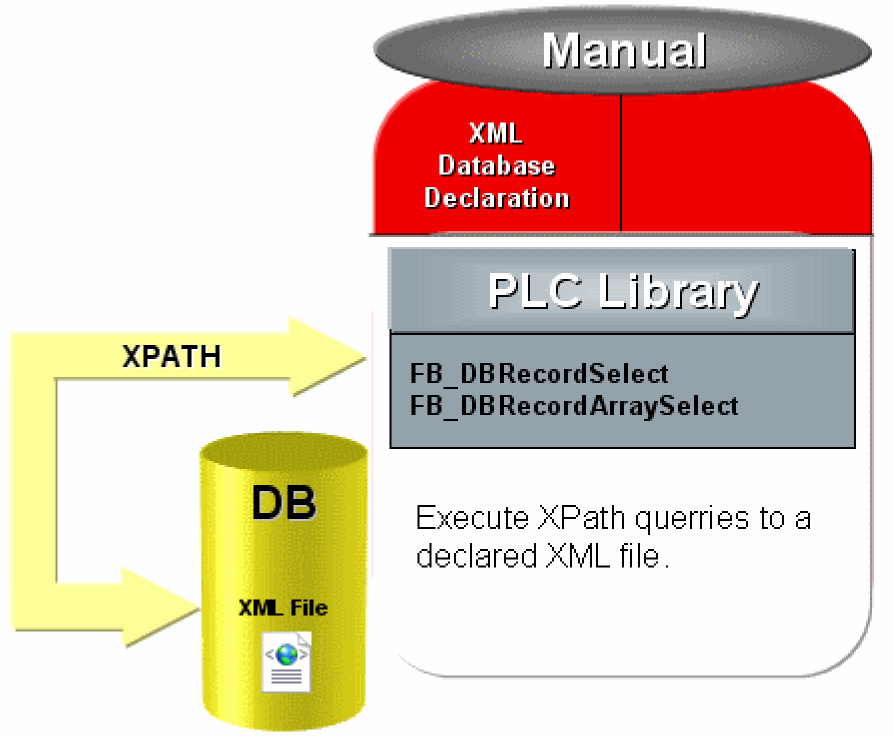
Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

7.2.7 XPath sample to illustrate the different SELECT types

The function block FB_DBRecordArraySelect/FB_DBRecordSelect can be used to issue XPath commands and read XML tags from any XML file. This sample illustrates reading of different entries from XML files via the TwinCAT Database Server. Individual tags, subtags and reading of attributes is supported, and these are displayed.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip



Database type used	XML
Compatible database types	XML
Function blocks used	FB_DBRecordArraySelect
Libraries to be integrated	Tc2_Database, Tc2_System, Tc2_Standard, Tc2_Utilities
Download file list	TcDBSrv_InfoSysSamples.tszip

Sample XML file (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>
```

ST_FactoryInfo structure

```
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
```

ST_Employee structure

```
TYPEST_Employee :
STRUCT
 sID          : T_MaxString;
 sName        : T_MaxString;
 sDepartment  : T_MaxString;
 sPosition    : T_MaxString;
 sHired       : T_MaxString;
END_STRUCT
END_TYPE
```

MAIN program

```
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
```

A positive edge at the variable "bStart" triggers issuing of the XPath commands and reading of the individual elements from the XML file. The results will then be in the variables "sFactoryName", "stFactoryInfo" and "aEmployees".

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

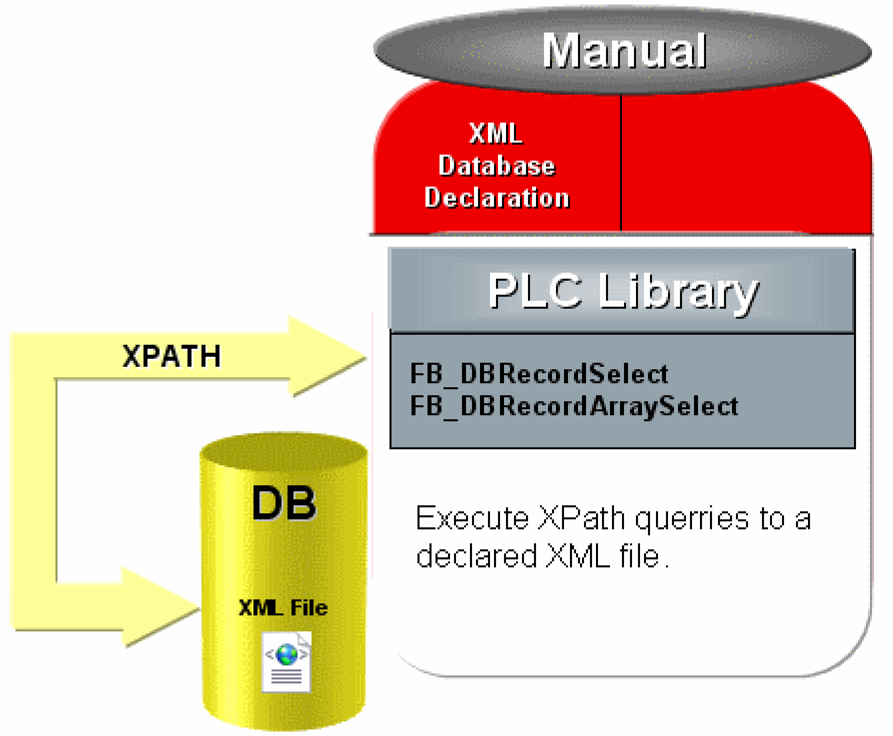
7.2.8 XPath sample with XML schema

The function blocks FB_DBRecordSelect or FB_DBRecordArraySelect can be used to issue XPath commands and to read XML tags, XML subtags or XML attributes from any XML file. If a suitable XML schema exists for the XML file to be read, the content of the tags or attributes is converted to the corresponding data types, as defined in the schema.

Further information about XML schemas can be found here: http://www.edition-w3.de/TR/2001/REC-xmlschema-0-20010502/

In this sample, FB_DBRecordArraySelect is used to read two different subtags from an XML file with corresponding XML schema.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip



Database type used	XML
Compatible database types	XML
Function blocks used	FB_DBRecordSelect
Libraries to be integrated	Tc2_Database, Tc2_System, Tc2_Standard, Tc2_Utilities
Download file list	TcDBSrv_InfoSysSamples.tszip, CurrentConfigDatabase.xml, PLC_Structs.xml, PLC_Structs.xsd

Sample XML file (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>
```

Corresponding 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>
```

Structure1 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
```

Structure2 ST_TestStruct2

```
TYPE ST_TestStruct2 :
STRUCT
    sSTRING: T_MaxString;
    bBOOL  : BOOL;
    nINT32 : DINT;
END_STRUCT
END_TYPE
```

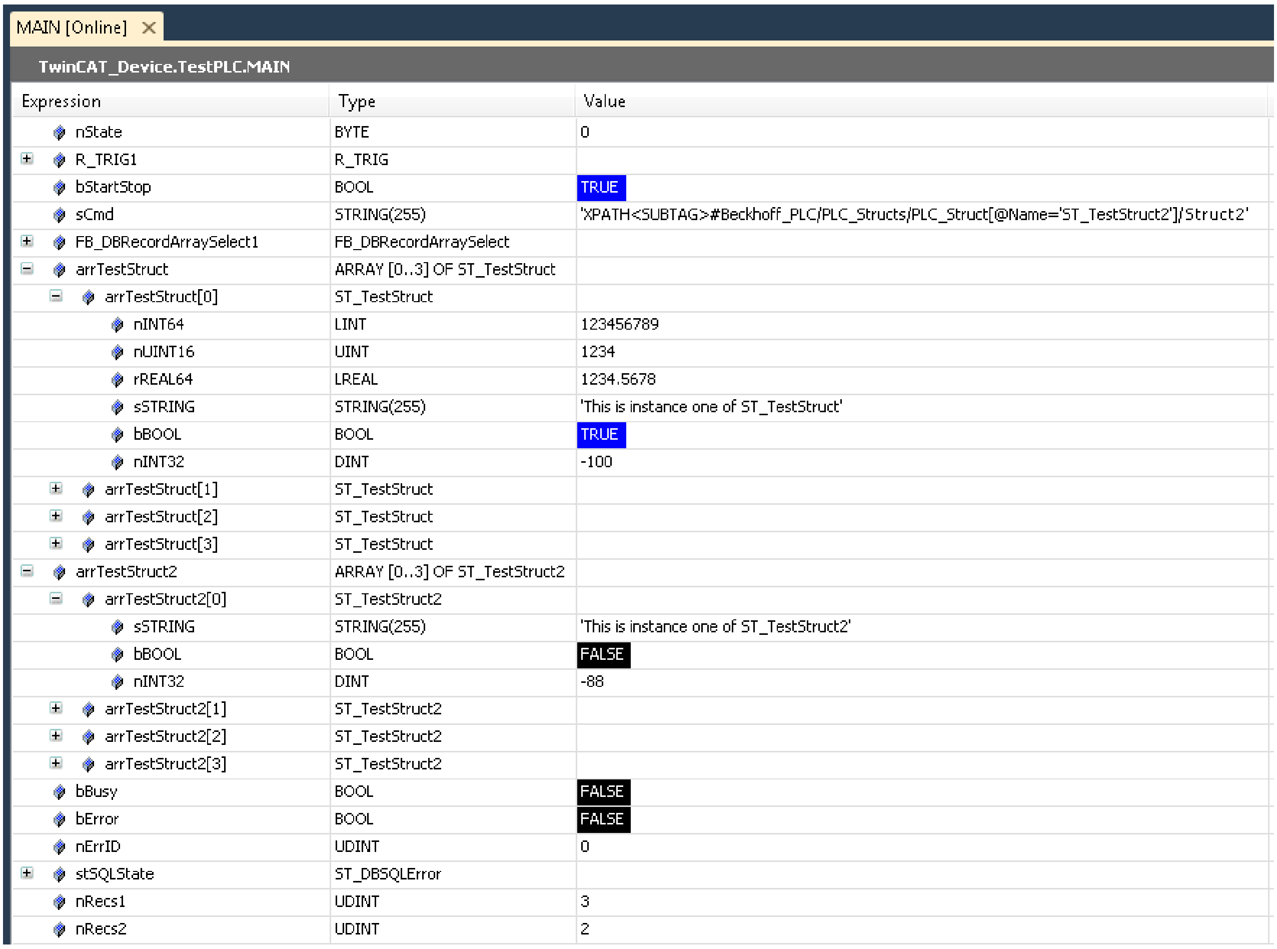
MAIN program

```
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
```

Reading is started with a positive edge at the toggle variable "bStartStop".



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

Requirements

Development environment	Target platform	PLC libraries to be linked
TwinCAT v3.0.0	PC or CX (x86)	Tc2_Database

8 Appendix

8.1 Error codes

8.1.1 Tc3_Database

8.1.1.1 PLC return values

The error output of all PLC blocks of the Tc3_Database.compiled library takes place via the I_TcResultEvent interfaces from the Tc3_Eventlogger.compiled library. This new interface structure enables a more detailed description of events, as well as classification.

```
Interface I_TcMessage [▶ 212]
    nEventId: UDINT;
    EventClass: GUID;
    eSeverity: TcEventSeverity [▶ 213];
    ipSourceInfo: I_TcSourceInfo;
```

nEventID: specific event code

EventClass: GUID

EventClassName: The corresponding event class name can be read using the RequestEventClassName method

eSeverity: events classification: from "info" to "critical error"

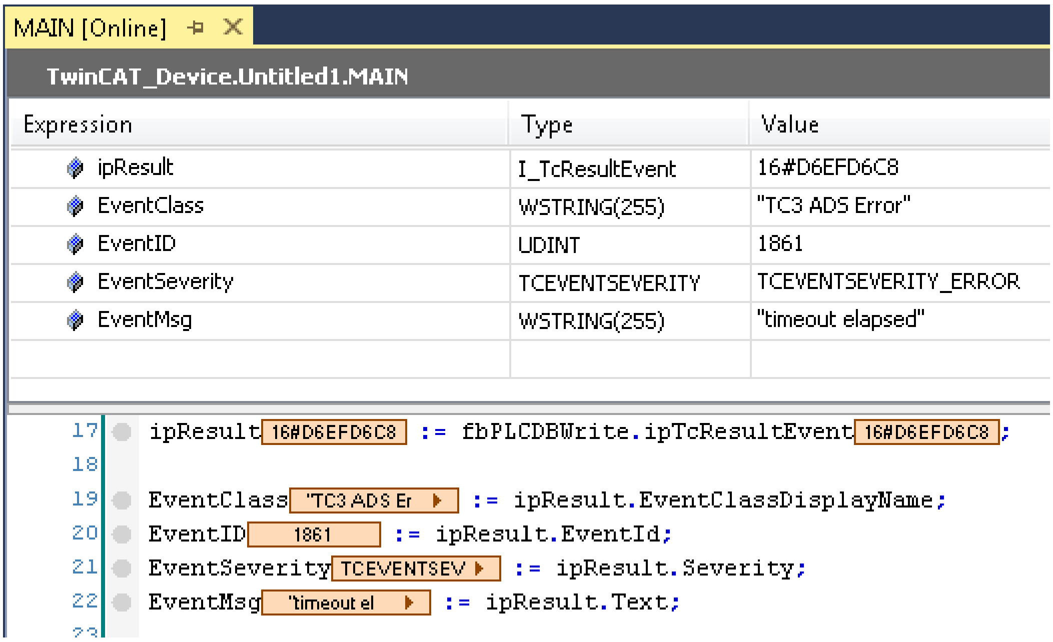
ipSourceInfo: path to the event location.

Text: description of the event in plain text can be read with the RequestEventText method

The following event classes can occur:

- **TC3 ADS Error**
  ADS errors that may occur during the communication with the TwinCAT Database Server.
- **TC3 Database Server Internal Error**
  Internal errors that may occur if the TwinCAT Database Server is configured incorrectly.
- **TC3 Database Server Database Error**
  Database errors that may occur during communication with the corresponding databases. The different database-specific error codes are mapped in a database error list. The database-specific codes are written into the ErrorLog, as required.
- **TC3 Database Server ADS Device Error**
  ADS error that may occur during internal communication with configured ADS devices.
- **TC3 Database Server NoSQL Error**
  Database error of a NoSQL database that occurred during communication with the corresponding databases.

8.1.1.2 ADS return codes



Expression	Type	Value
ipResult	I_TcResultEvent	16#D6EFD6C8
EventClass	WSTRING(255)	"TC3 ADS Error"
EventID	UDINT	1861
EventSeverity	TCEVENTSEVERITY	TCEVENTSEVERITY_ERROR
EventMsg	WSTRING(255)	"timeout elapsed"

```
ipResult 16#D6EFD6C8 := fbPLCDBWrite.ipTcResultEvent 16#D6EFD6C8;

EventClass 'TC3 ADS Er := ipResult.EventClassDisplayName;
EventID 1861 := ipResult.EventId;
EventSeverity TCEVENTSEV := ipResult.Severity;
EventMsg 'timeout el := ipResult.Text;
```

Global Error Codes

Hex	Dec	Description
0x0	0	no error
0x1	1	Internal error
0x2	2	No Rtime
0x3	3	Allocation locked memory error
0x4	4	Insert mailbox error
0x5	5	Wrong receive HMSG
0x6	6	target port not found
0x7	7	target machine not found
0x8	8	Unknown command ID
0x9	9	Bad task ID
0xA	10	No IO
0xB	11	Unknown ADS command
0xC	12	Win 32 error
0xD	13	Port not connected
0xE	14	Invalid ADS length
0xF	15	Invalid AMS Net ID
0x10	16	Low Installation level
0x11	17	No debug available
0x12	18	Port disabled
0x13	19	Port already connected
0x14	20	ADS Sync Win32 error
0x15	21	ADS Sync Timeout
0x16	22	ADS Sync AMS error
0x17	23	ADS Sync no index map
0x18	24	Invalid ADS port
0x19	25	No memory
0x1A	26	TCP send error
0x1B	27	Host unreachable
0x1C	28	Invalid AMS fragment

RTime Error Codes

Hex	Dec	Name	Description
0x1000	4096	RTERR_INTERNAL	Internal fatal error in the TwinCAT real-time system
0x1001	4097	RTERR_BADTIMERPERIODS	Timer value not vaild
0x1002	4098	RTERR_INVALIDTASKPTR	Task pointer has the invalid value ZERO
0x1003	4099	RTERR_INVALIDSTACKPTR	Task stack pointer has the invalid value ZERO
0x1004	4100	RTERR_PRIOEXISTS	The demand task priority is already assigned
0x1005	4101	RTERR_NOMORETCB	No more free TCB (Task Control Block) available. Maximum number of TCBs is 64
0x1006	4102	RTERR_NOMORESEMAS	No more free semaphores available. Maximum number of semaphores is 64
0x1007	4103	RTERR_NOMOREQUEUES	No more free queue available. Maximum number of queue is 64
0x100D	4109	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied
0x100E	4110	RTERR_EXTIRQNOTDEF	No external synchronization interrupt applied
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	The apply of the external synchronization interrupt failed
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in system BIOS
0x1019	4121	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension
0x101A	4122	RTERR_VMXENABLEFAILS	Enabling Intel VT-x fails

Router Error Codes

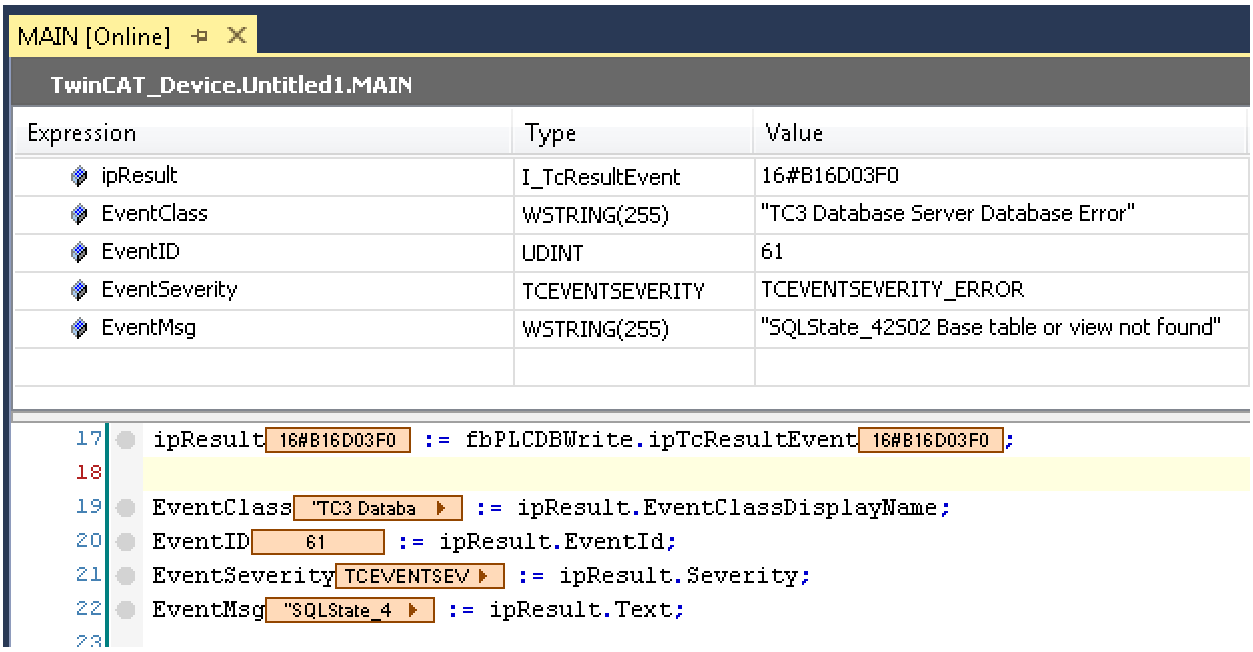
Hex	Dec	Name	Description
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	No locked memory can be allocated
0x501	1281	ROUTERERR_RESIZEMEMORY	The size of the router memory could not be changed
0x502	1282	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages. The current sent message was rejected
0x503	1283	ROUTERERR_DEBUGBOXFULL	The mailbox has reached the maximum number of possible messages. The sent message will not be displayed in the debug monitor
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	Unknown port type
0x505	1285	ROUTERERR_NOTINITIALIZED	Router is not initialized
0x506	1286	ROUTERERR_PORTALREADYINUSE	The desired port number is already assigned
0x507	1287	ROUTERERR_NOTREGISTERED	Port not registered
0x508	1288	ROUTERERR_NOMOREQUEUES	The maximum number of Ports reached
0x509	1289	ROUTERERR_INVALIDPORT	Invalid port
0x50A	1290	ROUTERERR_NOTACTIVATED	TwinCAT Router not active

General ADS Error Codes

Hex	Dec	Name	Description
0x700	1792	ADSERR_DEVICE_ERROR	General device error
0x701	1793	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by server
0x702	1794	ADSERR_DEVICE_INVALIDGRP	invalid index group
0x703	1795	ADSERR_DEVICE_INVALIDOFFSET	invalid index offset
0x704	1796	ADSERR_DEVICE_INVALIDACCESS	reading/writing not permitted
0x705	1797	ADSERR_DEVICE_INVALIDSIZE	parameter size not correct
0x706	1798	ADSERR_DEVICE_INVALIDDATA	invalid parameter value(s)
0x707	1799	ADSERR_DEVICE_NOTREADY	device is not in a ready state
0x708	1800	ADSERR_DEVICE_BUSY	device is busy
0x709	1801	ADSERR_DEVICE_INVALIDCONTEXT	invalid context (must be in Windows)
0x70A	1802	ADSERR_DEVICE_NOMEMORY	out of memory
0x70B	1803	ADSERR_DEVICE_INVALIDPARM	invalid parameter value(s)
0x70C	1804	ADSERR_DEVICE_NOTFOUND	not found (files, ...)
0x70D	1805	ADSERR_DEVICE_SYNTAX	syntax error in command or file
0x70E	1806	ADSERR_DEVICE_INCOMPATIBLE	objects do not match
0x70F	1807	ADSERR_DEVICE_EXISTS	object already exists
0x710	1808	ADSERR_DEVICE_SYMBOLNOTFOUND	symbol not found
0x711	1809	ADSERR_DEVICE_SYMBOLVERSIONINVAL	symbol version invalid
0x712	1810	ADSERR_DEVICE_INVALIDSTATE	server is in invalid state
0x713	1811	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported
0x714	1812	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid
0x715	1813	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered
0x716	1814	ADSERR_DEVICE_NOMOREHDLS	no more notification handles
0x717	1815	ADSERR_DEVICE_INVALIDWATCHSIZE	size for watch too big
0x718	1816	ADSERR_DEVICE_NOTINIT	device not initialized
0x719	1817	ADSERR_DEVICE_TIMEOUT	device has a timeout
0x71A	1818	ADSERR_DEVICE_NOINTERFACE	query interface failed
0x71B	1819	ADSERR_DEVICE_INVALIDINTERFACE	wrong interface required
0x71C	1820	ADSERR_DEVICE_INVALIDCLSID	class ID is invalid
0x71D	1821	ADSERR_DEVICE_INVALIDOBJID	object ID is invalid
0x71E	1822	ADSERR_DEVICE_PENDING	request is pending
0x71F	1823	ADSERR_DEVICE_ABORTED	request is aborted
0x720	1824	ADSERR_DEVICE_WARNING	signal warning
0x721	1825	ADSERR_DEVICE_INVALIDARRAYIDX	invalid array index
0x722	1826	ADSERR_DEVICE_SYMBOLNOTACTIVE	symbol not active
0x723	1827	ADSERR_DEVICE_ACCESSDENIED	access denied
0x724	1828	ADSERR_DEVICE_LICENSENOTFOUND	missing license
0x725	1829	ADSERR_DEVICE_LICENSEEXPIRED	license expired
0x726	1830	ADSERR_DEVICE_LICENSEEXCEEDED	license exceeded
0x727	1831	ADSERR_DEVICE_LICENSEINVALID	license invalid
0x728	1832	ADSERR_DEVICE_LICENSESYSTEMID	license invalid system id
0x729	1833	ADSERR_DEVICE_LICENSENOTIMELIMIT	license not time limited
0x72A	1834	ADSERR_DEVICE_LICENSEFUTUREISSUE	license issue time in the future
0x72B	1835	ADSERR_DEVICE_LICENSETIMETOLONG	license time period to long
0x72c	1836	ADSERR_DEVICE_EXCEPTION	exception occured during system start
0x72D	1837	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice
0x72E	1838	ADSERR_DEVICE_SIGNATUREINVALID	invalid signature
0x72F	1839	ADSERR_DEVICE_CERTIFICATEINVALID	public key certificate
0x740	1856	ADSERR_CLIENT_ERROR	Error class <client error>
0x741	1857	ADSERR_CLIENT_INVALIDPARM	invalid parameter at service
0x742	1858	ADSERR_CLIENT_LISTEMPTY	polling list is empty
0x743	1859	ADSERR_CLIENT_VARUSED	var connection already in use
0x744	1860	ADSERR_CLIENT_DUPLINVOKEID	invoke ID in use
0x745	1861	ADSERR_CLIENT_SYNCTIMEOUT	timeout elapsed
0x746	1862	ADSERR_CLIENT_W32ERROR	error in win32 subsystem
0x747	1863	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value

Hex	Dec	Name	Description
0x748	1864	ADSERR_CLIENT_PORTNOTOPEN	ads-port not opened
0x750	1872	ADSERR_CLIENT_NOAMSADDR	internal error in ads sync
0x751	1873	ADSERR_CLIENT_SYNCINTERNAL	hash table overflow
0x752	1874	ADSERR_CLIENT_ADDHASH	key not found in hash
0x753	1875	ADSERR_CLIENT_REMOVEHASH	no more symbols in cache
0x754	1876	ADSERR_CLIENT_NOMORESYM	invalid response received
0x755	1877	ADSERR_CLIENT_SYNCRESINVALID	sync port is locked

8.1.1.3 Database return codes



Expression	Type	Value
ipResult	I_TcResultEvent	16#B16D03F0
EventClass	WSTRING(255)	"TC3 Database Server Database Error"
EventID	UDINT	61
EventSeverity	TCEVENTSEVERITY	TCEVENTSEVERITY_ERROR
EventMsg	WSTRING(255)	"SQLState_42S02 Base table or view not found"

```
ipResult 16#B16D03F0 := fbPLCDBWrite.ipTcResultEvent 16#B16D03F0 ;

EventClass 'TC3 Databa := ipResult.EventClassDisplayName;
EventID 61 := ipResult.EventId;
EventSeverity TCEVENTSEV := ipResult.Severity;
EventMsg 'SQLState_4 := ipResult.Text;
```

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 row set
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	Date/time 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

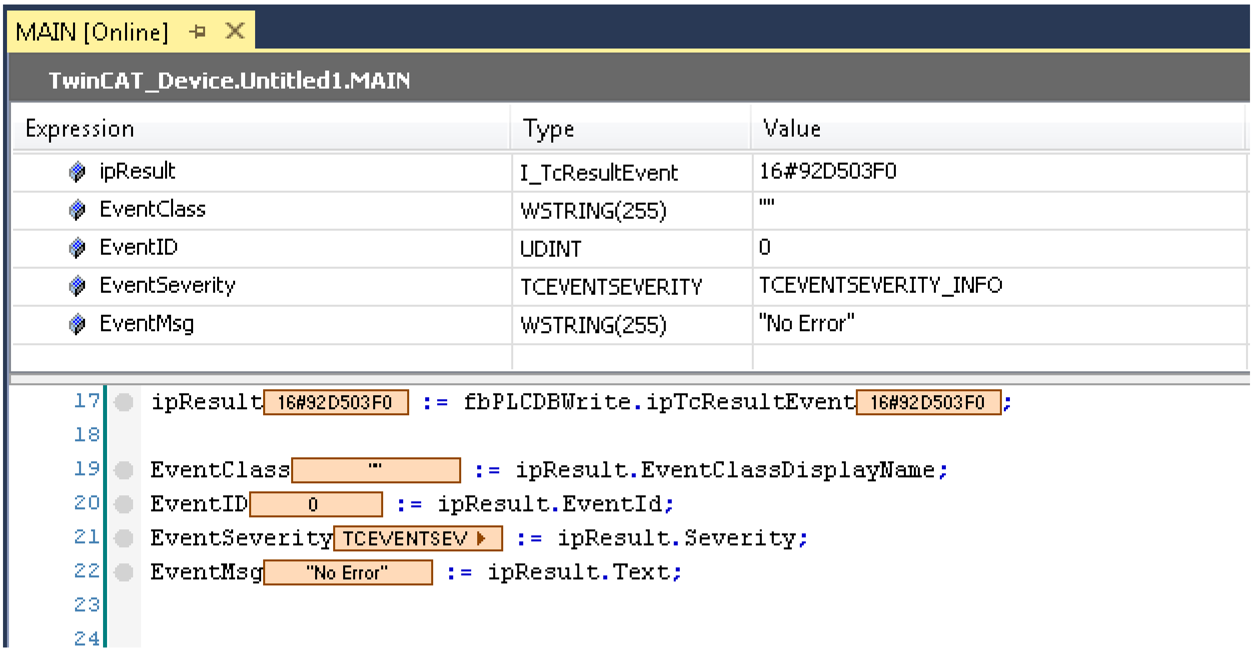
44	DB_SQLState_23000	Integrity constraint violation
45	DB_SQLState_24000	Invalid cursor state
46	DB_SQLState_25000	Invalid transaction state
47	DB_SQLState_25S01	Transaction state
48	DB_SQLState_25S02	Transaction is still active
49	DB_SQLState_25S03	Transaction is rolled back
50	DB_SQLState_28000	Invalid 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 cancelled
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 database return 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



TwinCAT_Device.Untitled1.MAIN

Expression	Type	Value
ipResult	I_TcResultEvent	16#92D503F0
EventClass	WSTRING(255)	""
EventID	UDINT	0
EventSeverity	TCEVENTSEVERITY	TCEVENTSEVERITY_INFO
EventMsg	WSTRING(255)	"No Error"

```
ipResult 16#92D503F0 := fbPLCDBWrite.ipTcResultEvent 16#92D503F0 ;

EventClass "" := ipResult.EventClassDisplayName;
EventID 0 := ipResult.EventId;
EventSeverity TCEVENTSEV := ipResult.Severity;
EventMsg "No Error" := ipResult.Text;
```

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

Error codes: 0x000 [▶ 381]..., 0x500 [▶ 381]..., 0x700 [▶ 382]..., 0x1000 [▶ 384]...

HRESULT

When output in HRESULT format, the ADS return codes are preceded by the high word 16#9811. The error 'Destination port not found' is then output as 16#9811_0006, for example.

Global Error Codes

Hex	Dec	Description
0x0	0	no error
0x1	1	Internal error
0x2	2	No Rtime
0x3	3	Allocation locked memory error
0x4	4	Insert mailbox error
0x5	5	Wrong receive HMSG
0x6	6	target port not found
0x7	7	target machine not found
0x8	8	Unknown command ID
0x9	9	Bad task ID
0xA	10	No IO
0xB	11	Unknown ADS command
0xC	12	Win 32 error
0xD	13	Port not connected
0xE	14	Invalid ADS length
0xF	15	Invalid AMS Net ID
0x10	16	Low Installation level
0x11	17	No debug available
0x12	18	Port disabled
0x13	19	Port already connected
0x14	20	ADS Sync Win32 error
0x15	21	ADS Sync Timeout
0x16	22	ADS Sync AMS error
0x17	23	ADS Sync no index map
0x18	24	Invalid ADS port
0x19	25	No memory
0x1A	26	TCP send error
0x1B	27	Host unreachable
0x1C	28	Invalid AMS fragment

Router Error Codes

Hex	Dec	Name	Description
0x500	1280	ROUTERERR_NOLOCKEDMEMORY	No locked memory can be allocated
0x501	1281	ROUTERERR_RESIZEMEMORY	The size of the router memory could not be changed
0x502	1282	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages. The current sent message was rejected
0x503	1283	ROUTERERR_DEBUGBOXFULL	The mailbox has reached the maximum number of possible messages. The sent message will not be displayed in the debug monitor
0x504	1284	ROUTERERR_UNKNOWNPORTTYPE	Unknown port type
0x505	1285	ROUTERERR_NOTINITIALIZED	Router is not initialized
0x506	1286	ROUTERERR_PORTALREADYINUSE	The desired port number is already assigned
0x507	1287	ROUTERERR_NOTREGISTERED	Port not registered
0x508	1288	ROUTERERR_NOMOREQUEUES	The maximum number of Ports reached
0x509	1289	ROUTERERR_INVALIDPORT	Invalid port
0x50A	1290	ROUTERERR_NOTACTIVATED	TwinCAT Router not active

General ADS Error Codes

Hex	Dec	Name	Description
0x700	1792	ADSERR_DEVICE_ERROR	General device error
0x701	1793	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by server
0x702	1794	ADSERR_DEVICE_INVALIDGRP	invalid index group
0x703	1795	ADSERR_DEVICE_INVALIDOFFSET	invalid index offset
0x704	1796	ADSERR_DEVICE_INVALIDACCESS	reading/writing not permitted
0x705	1797	ADSERR_DEVICE_INVALIDSIZE	parameter size not correct
0x706	1798	ADSERR_DEVICE_INVALIDDATA	invalid parameter value(s)
0x707	1799	ADSERR_DEVICE_NOTREADY	device is not in a ready state
0x708	1800	ADSERR_DEVICE_BUSY	device is busy
0x709	1801	ADSERR_DEVICE_INVALIDCONTEXT	invalid context (must be in Windows)
0x70A	1802	ADSERR_DEVICE_NOMEMORY	out of memory
0x70B	1803	ADSERR_DEVICE_INVALIDPARM	invalid parameter value(s)
0x70C	1804	ADSERR_DEVICE_NOTFOUND	not found (files, ...)
0x70D	1805	ADSERR_DEVICE_SYNTAX	syntax error in command or file
0x70E	1806	ADSERR_DEVICE_INCOMPATIBLE	objects do not match
0x70F	1807	ADSERR_DEVICE_EXISTS	object already exists
0x710	1808	ADSERR_DEVICE_SYMBOLNOTFOUND	symbol not found
0x711	1809	ADSERR_DEVICE_SYMBOLVERSIONINVAL	symbol version invalid
0x712	1810	ADSERR_DEVICE_INVALIDSTATE	server is in invalid state
0x713	1811	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported
0x714	1812	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid
0x715	1813	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered
0x716	1814	ADSERR_DEVICE_NOMOREHDLS	no more notification handles
0x717	1815	ADSERR_DEVICE_INVALIDWATCHSIZE	size for watch too big
0x718	1816	ADSERR_DEVICE_NOTINIT	device not initialized
0x719	1817	ADSERR_DEVICE_TIMEOUT	device has a timeout
0x71A	1818	ADSERR_DEVICE_NOINTERFACE	query interface failed
0x71B	1819	ADSERR_DEVICE_INVALIDINTERFACE	wrong interface required
0x71C	1820	ADSERR_DEVICE_INVALIDCLSID	class ID is invalid
0x71D	1821	ADSERR_DEVICE_INVALIDOBJID	object ID is invalid
0x71E	1822	ADSERR_DEVICE_PENDING	request is pending
0x71F	1823	ADSERR_DEVICE_ABORTED	request is aborted
0x720	1824	ADSERR_DEVICE_WARNING	signal warning
0x721	1825	ADSERR_DEVICE_INVALIDARRAYIDX	invalid array index
0x722	1826	ADSERR_DEVICE_SYMBOLNOTACTIVE	symbol not active
0x723	1827	ADSERR_DEVICE_ACCESSDENIED	access denied
0x724	1828	ADSERR_DEVICE_LICENSENOTFOUND	missing license
0x725	1829	ADSERR_DEVICE_LICENSEEXPIRED	license expired
0x726	1830	ADSERR_DEVICE_LICENSEEXCEEDED	license exceeded
0x727	1831	ADSERR_DEVICE_LICENSEINVALID	license invalid
0x728	1832	ADSERR_DEVICE_LICENSESYSTEMID	license invalid system id
0x729	1833	ADSERR_DEVICE_LICENSENOTIMELIMIT	license not time limited
0x72A	1834	ADSERR_DEVICE_LICENSEFUTUREISSUE	license issue time in the future
0x72B	1835	ADSERR_DEVICE_LICENSETIMETOLONG	license time period to long
0x72c	1836	ADSERR_DEVICE_EXCEPTION	exception occured during system start
0x72D	1837	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice
0x72E	1838	ADSERR_DEVICE_SIGNATUREINVALID	invalid signature
0x72F	1839	ADSERR_DEVICE_CERTIFICATEINVALID	public key certificate
0x740	1856	ADSERR_CLIENT_ERROR	Error class <client error>
0x741	1857	ADSERR_CLIENT_INVALIDPARM	invalid parameter at service
0x742	1858	ADSERR_CLIENT_LISTEMPTY	polling list is empty
0x743	1859	ADSERR_CLIENT_VARUSED	var connection already in use
0x744	1860	ADSERR_CLIENT_DUPLINVOKEID	invoke ID in use
0x745	1861	ADSERR_CLIENT_SYNCTIMEOUT	timeout elapsed
0x746	1862	ADSERR_CLIENT_W32ERROR	error in win32 subsystem
0x747	1863	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value

Hex	Dec	Name	Description
0x748	1864	ADSERR_CLIENT_PORTNOTOPEN	ads-port not opened
0x750	1872	ADSERR_CLIENT_NOAMSADDR	internal error in ads sync
0x751	1873	ADSERR_CLIENT_SYNCINTERNAL	hash table overflow
0x752	1874	ADSERR_CLIENT_ADDHASH	key not found in hash
0x753	1875	ADSERR_CLIENT_REMOVEHASH	no more symbols in cache
0x754	1876	ADSERR_CLIENT_NOMORESYM	invalid response received
0x755	1877	ADSERR_CLIENT_SYNCRESINVALID	sync port is locked

RTime Error Codes

Hex	Dec	Name	Description
0x1000	4096	RTERR_INTERNAL	Internal fatal error in the TwinCAT real-time system
0x1001	4097	RTERR_BADTIMERPERIODS	Timer value not vaild
0x1002	4098	RTERR_INVALIDTASKPTR	Task pointer has the invalid value ZERO
0x1003	4099	RTERR_INVALIDSTACKPTR	Task stack pointer has the invalid value ZERO
0x1004	4100	RTERR_PRIOEXISTS	The demand task priority is already assigned
0x1005	4101	RTERR_NOMORETCB	No more free TCB (Task Control Block) available. Maximum number of TCBs is 64
0x1006	4102	RTERR_NOMORESEMAS	No more free semaphores available. Maximum number of semaphores is 64
0x1007	4103	RTERR_NOMOREQUEUES	No more free queue available. Maximum number of queue is 64
0x100D	4109	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied
0x100E	4110	RTERR_EXTIRQNOTDEF	No external synchronization interrupt applied
0x100F	4111	RTERR_EXTIRQINSTALLFAILED	The apply of the external synchronization interrupt failed
0x1010	4112	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported
0x1018	4120	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in system BIOS
0x1019	4121	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension
0x101A	4122	RTERR_VMXENABLEFAILS	Enabling Intel VT-x fails

TCP Winsock Error Codes

Hex	Dec	Description
0x274D	10061	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
0x2751	10065	No connection could be made because the target machine actively refused it. This error normally occurs when you try to connect to a service which is inactive on a different host - a service without a server application.
0x274C	10060	No route to a host. A socket operation was attempted to an unreachable host
		Further Winsock error codes: Win32 Error Codes

8.1.2.2 TwinCAT Database Server error codes overview

Code (hex)	Code (Dec)	Description
0x0001 + ADS error code	65537 - 131071	ADS error code from declared ADS device
0x00020001	131073	Microsoft SQL Compact database (error code)
0x00040001	262145	Microsoft SQL database (error code)
0x00080001	524289	Microsoft Access database (error code)
0x00100001	1048577	MySQL database (error code)
0x00200001	2097153	Oracle database (error code)
0x00400001	4194305	DB2 database (error code)
0x00800001	8388609	PostgreSQL database (error code)
0x01000001	16777217	Interbase/Firebird database (error code)
0x02000001	33554433	TwinCAT Database Server error code [▶ 391]
0x04000001	67108865	XML database (error code)
0x08000001	134217729	ASCII database (error code)

If one of the error codes mentioned above is issued at the "nErrID" output of a function block, an error has occurred during execution of an SQL statement. The SQL error code is then issued at the "sSQLState" output of the function block. The "sSQLState" output has the data type ST_DBSQLError [▶ 310]. For each database type individual error codes are output.

A list of SQLStates can be found under: http://msdn.microsoft.com/en-us/library/ms714687(VS.85).aspx (SQLStates)

Database type	Error code reference
Microsoft SQL Compact database	http://technet.microsoft.com/en-us/library/ms171788.aspx / OleDB_Errorcodes.htm [▶ 386]
Microsoft SQL database	OleDB_Errorcodes.htm [▶ 386]
Microsoft Access database	OleDB_Errorcodes.htm [▶ 386]
MySQL database	http://dev.mysql.com/doc/refman/5.0/en/error-handling.html
Oracle database	http://www.ora-code.com
DB2 database	http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z9.doc.codes/src/tpc/db2z_n.htm
PostgreSQL database	http://www.postgresql.org/docs/current/static/errcodes-appendix.html
Interbase/Firebird database	http://www.firebirdsql.org/file/documentation/reference_manuals/reference_material/Firebird-2.1-ErrorCodes.pdf
XML database	TcDBServer_XML_Errorcodes.htm [▶ 391]
ASCII database	TcDBServer_ASCII_Errorcodes.htm [▶ 391]

8.1.2.3 OleDB error codes

Value	Description
0x80040E00	The accessor is invalid.
0x80040E01	It was not possible to insert a row into the row set, because the maximum number of active rows for the provider would have been exceeded.
0x80040E02	The accessor is write-protected. The procedure has failed.
0x80040E03	Values violate the database schema.
0x80040E04	The row handle is invalid.
0x80040E05	The object was open.
0x80040E06	Invalid chapter
0x80040E07	A literal value in the command could not be converted to the correct type for a reason other than data overflow.
0x80040E08	Invalid binding information
0x80040E09	Permission denied
0x80040E0A	The specified column contains no bookmarks or chapters.
0x80040E0B	Some cost limitations were rejected.
0x80040E0C	No command was specified for the command object.
0x80040E0D	No query plan was found within the specified cost limitation.
0x80040E0E	Invalid bookmark
0x80040E0F	Invalid lock mode
0x80040E10	No value was specified for at least one of the required parameters.
0x80040E11	Invalid column ID
0x80040E12	Invalid quota
0x80040E13	Invalid value
0x80040E14	The command contained at least one error.
0x80040E15	The currently executed command cannot be aborted.
0x80040E16	The provider offers no support for the specified dialect.
0x80040E17	A data source with the specified name already exists.
0x80040E18	The row set was created via a live datastream and cannot be restarted.
0x80040E19	In the current range no key matches the described characteristics.
0x80040E1B	The provider is unable to determine the identity for the newly added rows.
0x80040E1A	The ownership of this structure was transferred to the provider.
0x80040E1C	Non-zero weighting values are not supported as target information. The target was therefore rejected. The current target was not changed.
0x80040E1D	The requested conversion is not supported.
0x80040E1E	RowsOffset leads to position after the end of the row set, irrespective of the specified cRows value. cRowsObtained is 0.
0x80040E20	The provider has called an IRowsetNotify method in the consumer and has not yet received a return from the method.
0x80040E21	Error
0x80040E22	A non-zero controlling IUnknown object was specified, and the currently created object does not support aggregation.
0x80040E23	The current row was deleted.
0x80040E24	The row set does not support backward calls.
0x80040E25	All HROW objects have to be released before new HROW objects can be received.
0x80040E26	A specified memory flag was not supported.
0x80040E27	The comparison operator was invalid.
0x80040E28	The specified status flag was neither DBCOLUMNSTATUS_OK nor DBCOLUMNSTATUS_ISNULL.
0x80040E29	The row set cannot be processed backwards.
0x80040E2A	Invalid range handle.

Value	Description
0x80040E2B	The specified row set was not adjacent to the rows of the specified monitoring range, and there was no overlap.
0x80040E2C	A transition from ALL* to MOVE* or EXTEND* was specified.
0x80040E2D	The specified range is not a valid subrange of the range identified by the specified monitoring range handle.
0x80040E2E	The provider does not support commands with several statements.
0x80040E2F	A specified value violated the integrity restrictions for a column or table.
0x80040E30	The specified type name was not recognized.
0x80040E31	Execution was aborted, since no further resources were available. No results were returned.
0x80040E32	A command object with a command hierarchy containing at least one row set could not be cloned.
0x80040E33	The current structure cannot be shown as text.
0x80040E34	The specified index already exists.
0x80040E35	The specified index does not exist.
0x80040E36	The specified index was used.
0x80040E37	The specified table does not exist.
0x80040E38	The row set has uses fully parallelism, and the value of a column was changed since the last read operation.
0x80040E39	Errors were found during copying.
0x80040E3A	A precision statement was invalid.
0x80040E3B	A specified decimal value was invalid.
0x80040E3C	Invalid table ID.
0x80040E3D	A specified type was invalid.
0x80040E3E	A column ID occurred several times in the specification.
0x80040E3F	The specified table already exists.
0x80040E40	The specified table was used.
0x80040E41	The specified range schema ID was not supported.
0x80040E42	The specified record number is invalid.
0x80040E43	No matching row could be found, despite the fact that the bookmark formatting was valid.
0x80040E44	The value of a property was invalid.
0x80040E45	The row set was not subdivided into chapters.
0x80040E46	Invalid accessor
0x80040E47	Invalid memory flags
0x80040E48	Accessors for transfer as reference are not supported by this provider.
0x80040E49	NULL accessors are not supported by this provider.
0x80040E4A	The command was not prepared.
0x80040E4B	The specified accessor was not a parameter accessor.
0x80040E4C	The specified accessor was write-protected.
0x80040E4D	Error during authentication.
0x80040E4E	The change was aborted during the notification; no columns were modified.
0x80040E4F	The row set consisted of a chapter, but the chapter was not enabled.
0x80040E50	Invalid source handle
0x80040E51	The provider is unable to derive parameter information, and SetParameterInfo was not called.
0x80040E52	The data source object is already initialized.
0x80040E53	The provider does not support this method.
0x80040E54	The number of rows with pending modifications exceeds the specified limit.
0x80040E55	The specified column did not exist.

Value	Description
0x80040E56	Changes are pending in a row with a reference counter of zero.
0x80040E57	A literal value in the command let to a range violation for the type of the assigned column.
0x80040E58	The transferred HRESULT value was invalid.
0x80040E59	The transferred LookupID value was invalid.
0x80040E5A	The transferred DynamicErrorID value was invalid.
0x80040E5B	No visible data for a newly added row that has not yet been updated can be retrieved.
0x80040E5C	Invalid conversion flag
0x80040E5D	The specified parameter name was not recognized.
0x80040E5E	Several memory objects cannot be open simultaneously.
0x80040E5F	The requested filter could not be opened.
0x80040E60	The requested sequence could not be opened.
0x80040E65	The transferred columnID value was invalid.
0x80040E67	The transferred command has no DBID value.
0x80040E68	The transferred DBID value already exists.
0x80040E69	The maximum number of session objects supported by this provider has already been reached. The consumer must release at least one current session object, before a new session object can be retrieved.
0x80040E72	The index ID is invalid.
0x80040E73	The specified initialization character sequence does not match the specification.
0x80040E74	The OLE DB master enumerator has not returned any providers that match the requested SOURCES_TYPE value.
0x80040E75	The initialization character sequence indicates a provider that does not match the currently active provider.
0x80040E76	The specified DBID value is invalid.
0x80040E6A	Invalid value for trust recipient.
0x80040E6B	The trust recipient is not intended for the current data source.
0x80040E6C	The trust recipient offers no support for memberships/list.
0x80040E6D	The object is invalid, or the provider is unknown.
0x80040E6E	The object has no owner.
0x80040E6F	The transferred access entry list is invalid.
0x80040E70	The trust recipient transferred as owner is invalid, or the provider is unknown.
0x80040E71	The permission transferred in the access entry list is invalid.
0x80040E77	The ConstraintType value was invalid or was not supported by the provider.
0x80040E78	The ConstraintType value was not DBCONSTRAINTTYPE_FOREIGNKEY, and cForeignKeyColumns was not zero.
0x80040E79	The Deferability value was invalid or was not supported by the provider.
0x80040E80	The MatchType value was invalid or was not supported by the provider.
0x80040E8A	The UpdateRule or DeleteRule value was invalid or was not supported by the provider.
0x80040E8B	Invalid restriction ID.
0x80040E8C	The dwFlags value was invalid.
0x80040E8D	The rguidColumnType value pointed to a GUID that does not match the object type of this column, or this column was not specified.
0x80040E91	No source row exists.
0x80040E92	The OLE DB object represented by this URL is locked by at least one other process.
0x80040E93	The client requested an object type that is only for lists.
0x80040E94	The calling process requested write access for a write-protected object.
0x80040E95	The provider was unable to establish a connection with the server for this object.
0x80040E96	The provider was unable to establish a connection with the server for this object.

Value	Description
0x80040E97	Timeout during binding to the object
0x80040E98	The provider was unable to create an object with this URL, since an object named by this URL already exists.
0x80040E8E	The requested URL was outside the valid range.
0x80040E90	The column or restriction could not be deleted, since a dependent view or restriction refers to it.
0x80040E99	The restriction already exists.
0x80040E9A	The object cannot be created with this URL, since the server has insufficient physical memory.
0x00040EC0	During retrieval of the requested number of rows the total number of active rows supported by this row set was exceeded.
0x00040EC1	At least one column type is not compatible; conversion errors may occur during copying.
0x00040EC2	Information on the parameter type were disabled by the calling process.
0x00040EC3	The bookmark for a deleted or irrelevant row was skipped.
0x00040EC5	No further row sets are available.
0x00040EC6	Start or end of the row set or chapter reached.
0x00040EC7	The command was executed again by the provider.
0x00040EC8	The data buffer for the variable is full.
0x00040EC9	No further results are available.
0x00040ECA	The server is unable to cancel or downgrade a lockout until a transaction is complete.
0x00040ECB	The specified weighting value was not supported or exceeded the supported limit. The value was set to 0 or to the limit.
0x00040ECC	For this reason the consumer rejects further notification calls.
0x00040ECD	The input dialect was ignored, and the text was returned in another dialect.
0x00040ECE	The consumer rejects further notification calls for this phase.
0x00040ECF	For this reason the consumer rejects further notification calls.
0x00040ED0	The operation is processed asynchronously.
0x00040ED1	To reach the start of the row set, the provider has to execute the query again. Either the order of the columns has changed, or columns were added to the row set, or columns were removed from the row set.
0x00040ED2	The method had several errors. The errors were returned in the error array.
0x00040ED3	Invalid row handle
0x00040ED4	A specified HROW object referred to a permanently deleted row.
0x00040ED5	The provider was unable to trace all modifications. The client has to retrieve the data assigned the monitoring range again via another method.
0x00040ED6	The execution was terminated because no more resources were available. The results received up to this time were returned, but the execution cannot continue.
0x00040ED8	A lockout was upgraded relative to the specified value.
0x00040ED9	At least one property was changed according to the options permitted by the provider.
0x00040EDA	Error
0x00040EDB	A specified parameter was invalid.
0x00040EDC	Due to the update of this row several rows in the data source had to be updated.
0x00040ED7	The binding failed, since the provider was not able to meet all binding flags or properties.
0x00040EDD	The row contains no row-specific columns.

8.1.2.4 ASCII error codes

Value	Description
1	Function not available
2	Syntax error
3	File could not be opened.

8.1.2.5 XML error codes

Value	Description
1	Function not available
2	XML file could not be loaded
3	XML schema could not be loaded
4	Syntax error
5	Table could not be created
6	The INSERT VALUE list does not match the columns list
7	PLC structure is not large enough
8	XML file could not be created
9	XML database not found.
10	XML table not found

8.1.2.6 Internal error codes

Error code	Description
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 - frequently asked questions and answers

In this section frequently asked questions are answered, in order to facilitate your work with the TwinCAT Database Server.
If you have any further questions, contact please our Support.

1. What performance can be achieved with the TwinCAT Database Server? [▶ 392]

2. Are so-called stored procedures supported? [▶ 392]

3. Which database types does the TwinCAT Database Server support? [▶ 392]

4. Can old database server configurations still be used in later versions? [▶ 392]

5. How can individual variables be written into an existing database structure or read from it? [▶ 392]

6. Can several records be written into a database simultaneously? [▶ 392]

7. How can the TwinCAT Database Server be operated in a network? [▶ 392]

8. Which functions of the TwinCAT Database Server are supported by the database type "XML"? [▶ 392]

9. Which Visual Studio versions are currently supported by the database server configurator? [▶ 392]

What performance can be achieved with the TwinCAT Database Server?
This question cannot be answered in general terms. The performance that can be achieved depends on the hardware used, the actions to be execute (e.g. ring buffer logging) and the number of variables. Another key factor is the database type that is used.

Are so-called stored procedures supported?
Yes, the TwinCAT Database Server supports stored procedures. The function block FB_SQLStoredProcedure [▶ 191] is provided for this purpose in the PLC library. The SQL Query Editor [▶ 43] can be used to test stored procedures, and corresponding PLC code for the function block FB_SQLStoredProcedure can be generated. Not all databases support this function.

Which database types does the TwinCAT Database Server support?
Information on the supported databases can be found in section "Databases [▶ 118]".

Can old database server configurations still be used in later versions?
It goes without saying that we aim to ensure compatibility. This was also taken into account during major version upgrades or complete redesigns (e.g. old: 3.**0**.x, new: 3.**1**.x). Further details can be found in section "Compatibility [▶ 22]".

How can individual variables be written into an existing database structure or read from it?
The function block FB_SQLCommand [▶ 185] can be used to write individual variables into an existing database structure or read from it.

Can several records be written into a database simultaneously?
This depends on the database used. With a Microsoft SQL database this would be possible in conjunction with the function block FB_SQLCommand [▶ 185], since several SQL insert commands (separated by semicolon) can be transferred to the PLC function block.

How can the TwinCAT Database Server be operated in a network?
The TwinCAT Database Server can be used in a network in several ways. Further information on support network topologies can be found in section "Areas of application and network topologies [▶ 20]".

Which functions of the TwinCAT Database Server are supported by the database type "XML"?
The "XML" database type supports the full functionality of the TwinCAT Database Server, except for "stored procedures". The XML file can be used to communicate with any other database via SQL commands, and PLC values can be logged in the XML file with the cyclic write mode. In addition, it is possible to execute XPath commands and read the corresponding XML tags. Further information can be found in section "XML database [▶ 127]".

Which Visual Studio versions are currently supported by the database server configurator?

Currently the Visual Studio® versions 2013, 2015 and 2017 are supported with our configurator integration [▶ 25].

8.3 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff's branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages:
http://www.beckhoff.com

You will also find further documentation for Beckhoff components there.

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

Phone:	+49(0)5246/963-0
Fax:	+49(0)5246/963-198
e-mail:	info@beckhoff.com

Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline:	+49(0)5246/963-157
Fax:	+49(0)5246/963-9157
e-mail:	support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline:	+49(0)5246/963-460
Fax:	+49(0)5246/963-479
e-mail:	service@beckhoff.com