BECKHOFF New Automation Technology

Manual | EN

TS6420

TwinCAT 2 | DataBase Server

Supplement | Communication

2023-08-10 | Version: 1.2

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 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®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® 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, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

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

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are 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 technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

The TwinCAT Database Server enables data exchange between the TwinCAT System and different database systems. For smaller applications you can use the server over a configurator without influencing the existing program code. For complex tasks and a maximum of flexibility the Database Server offers a detailed library of PLC function blocks. Directly out of the PLC you can use SQL-commands like Insert and Select. If necessary, you can relieve the PLC by calling up Stored Procedures in the database. The transferred parameters from the PLC function block will be used from the database in connection with the stored procedure and results will be returned to the controller.

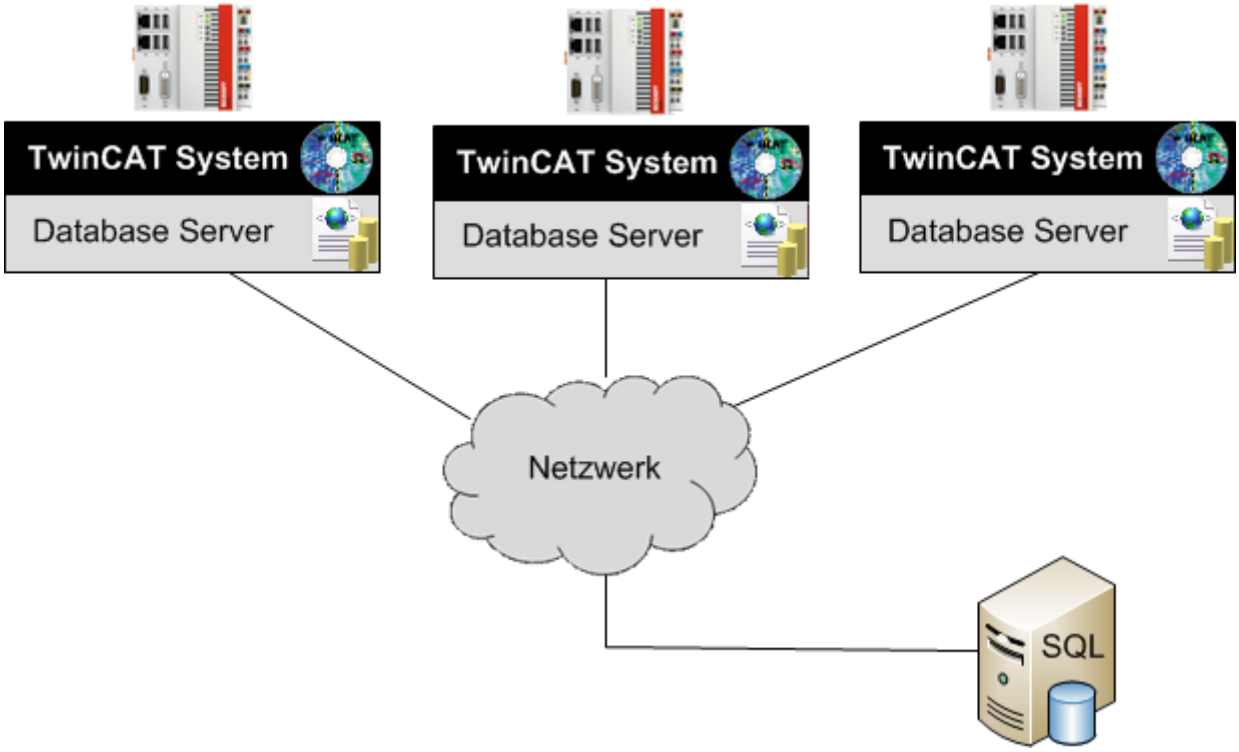
Now the TwinCAT Database Server supports eleven different database systems [▶ 35]: MS SQL, MS SQL Compact, MS Access, MySQL, PostgreSQL, DB2, Oracle, Interbase, Firebird, ASCII (e.g. .txt or .csv) as well as XML files.

Components

- TwinCAT Database Server [▶ 13]: a service which starts and stops along with TwinCAT. It is a connector between the TwinCAT system and the database.
- Configurator [▶ 15]: the TwinCAT Database Server configurator enables an easy visual setting of database parameters, which are necessary for the basic communication with the respective database
- PLC library [▶ 58]: the PLC library offers several function blocks to generate a database connection or a new chart. Furthermore you can write data into any chart structures with Insert-commands or read them through Select-commands. It is also possible to update or delete database entries. Stored procedures can be activated.

Functional principle

The Database Server communicates over ADS within the TwinCAT system. Outwards the server connects with the respective database. Possible network topologies can be found here [▶ 141].



3 System requirements

Requirements

Technical Data	TwinCAT Database Server
Targetsystem	Windows NT/2000/XP/Vista/7 PC (x86-compatible)
.NET Framework	.Net 2.0 oder higher
Min. TwinCAT-Version	2.10.0
Min. TwinCAT-Level	TwinCAT PLC

4 Installation

This part of the documentation gives a step-by-step explaination of the TwinCAT 3 Database Server setup process for Windows based operating systems. The following topics are part of this document:

- Starting the installation
- After the installation

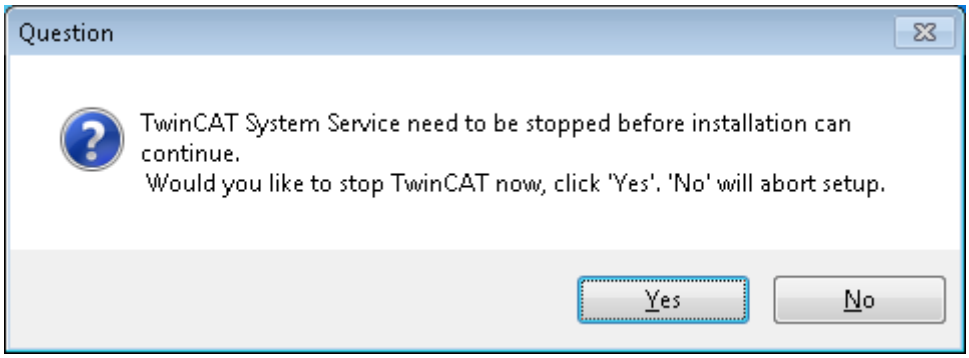
Starting the installation

Starting the installation

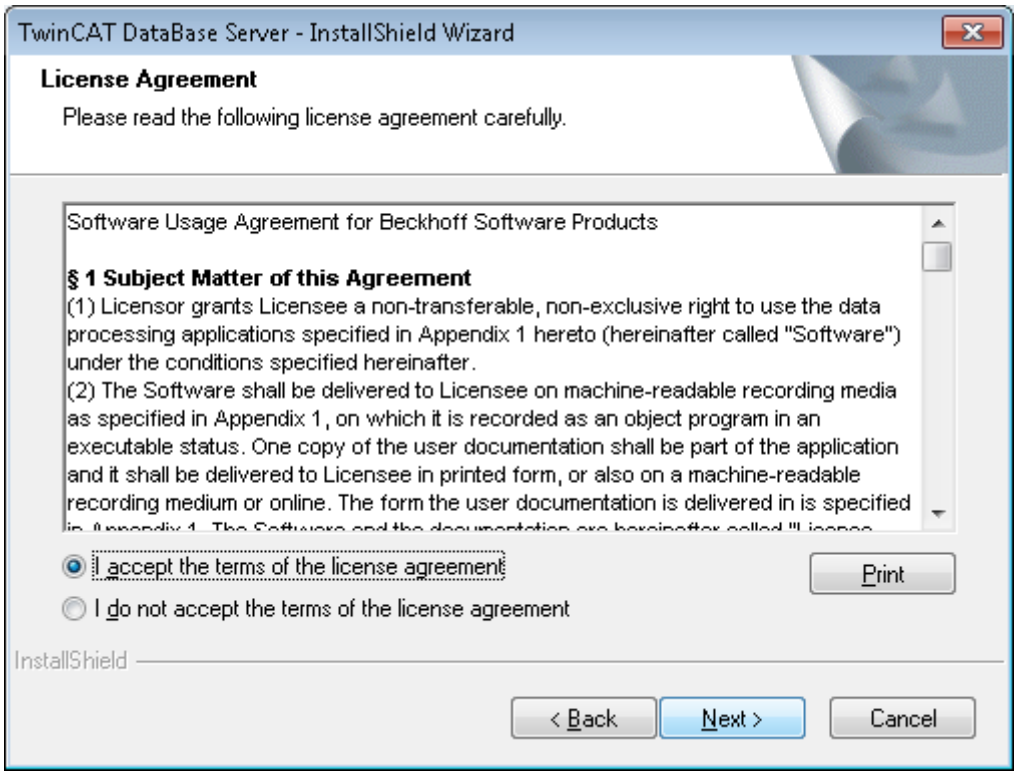
To install the Supplement, please perform the following steps:

Under Windows 32-bit/64-bit, please start the installation with "Run as Administrator" by right-clicking the setup file and selecting the corresponding option in the context menu.

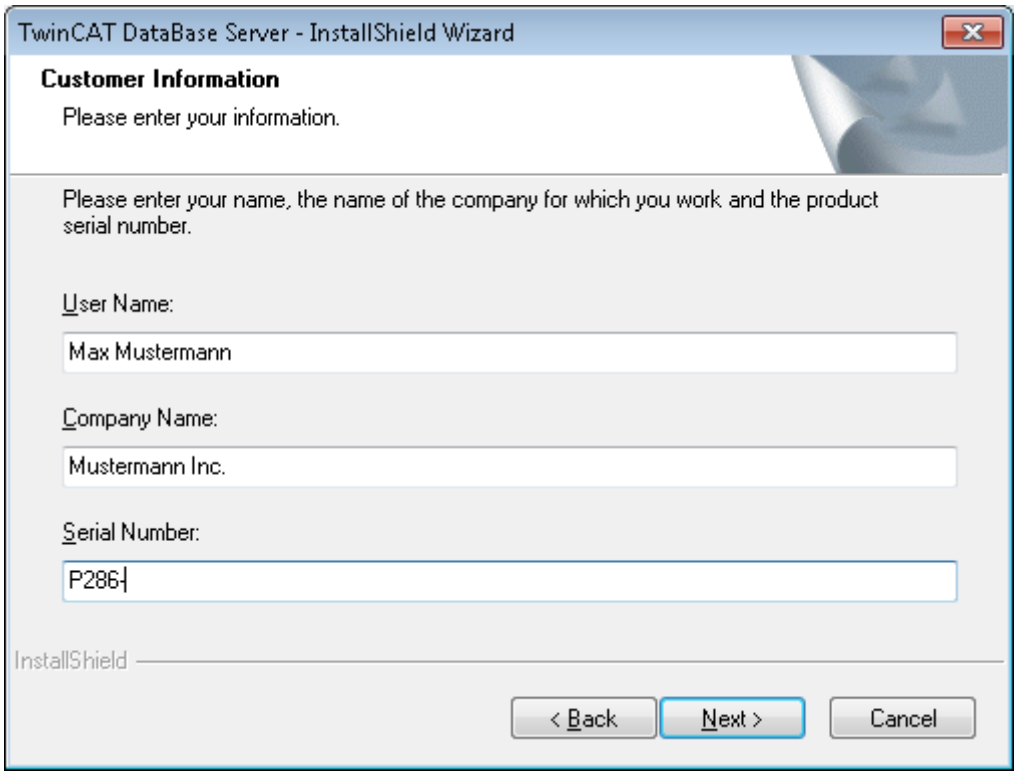
- Double-click the downloaded setup file **"TcDatabaseSrv.exe".**
- Select a language in which you want to install the software
- TwinCAT system must be stopped before proceeding with installation. Select "Yes" to stop TwinCAT system service



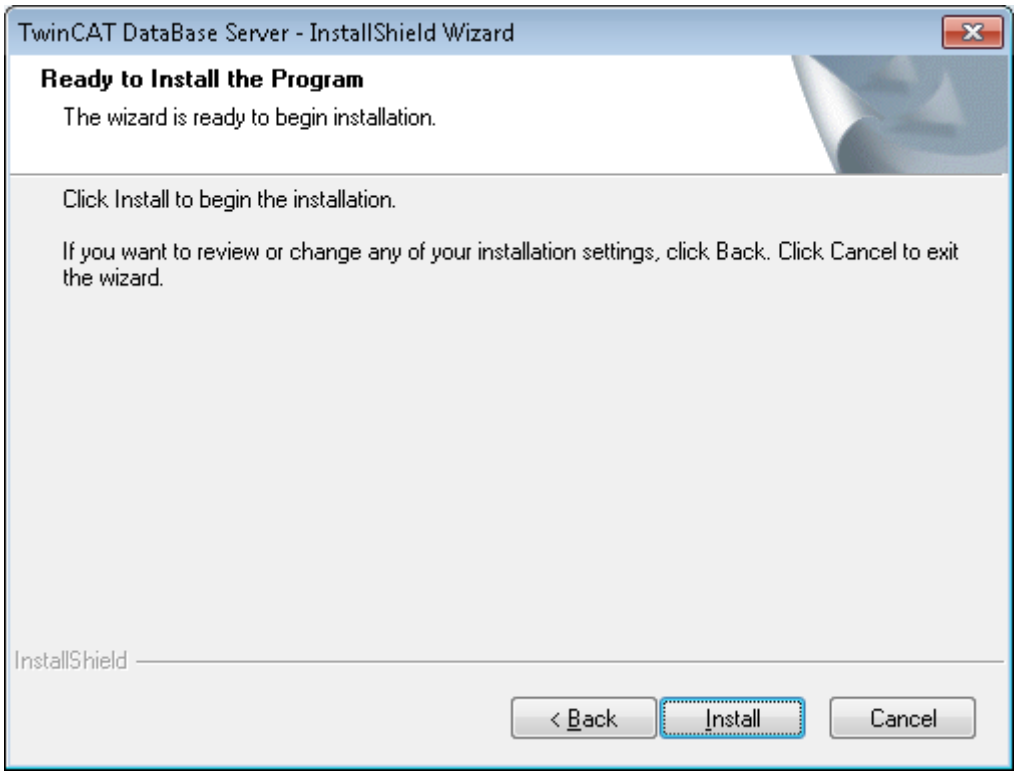
- Click on "Next" and accept the license agreement



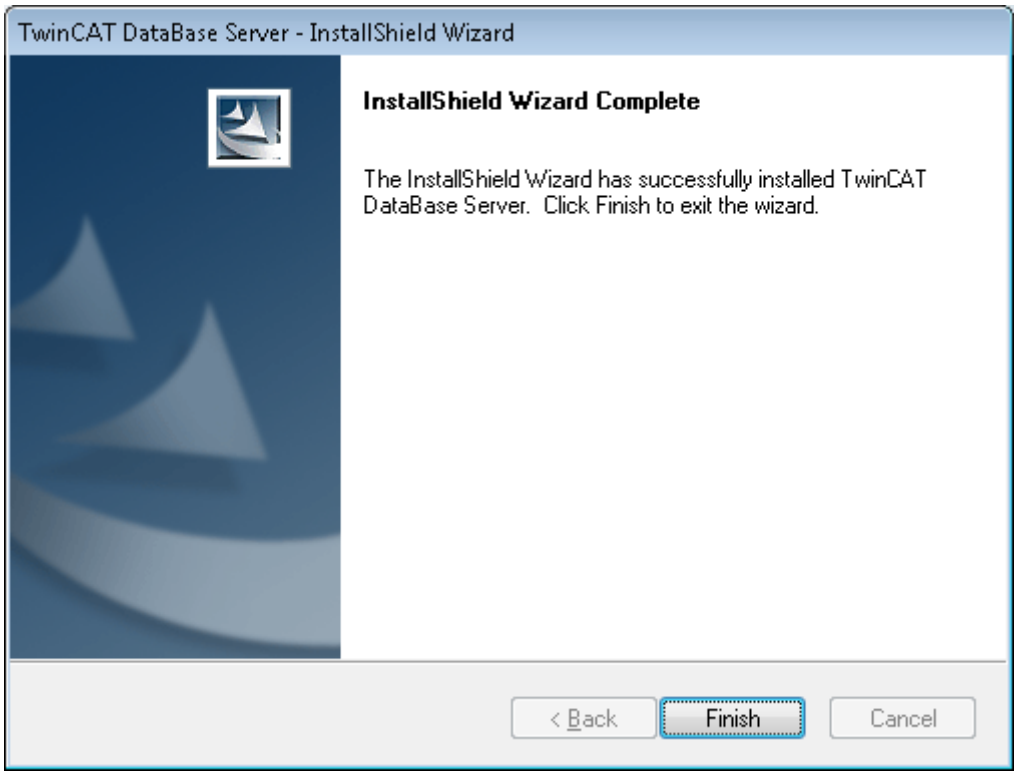
- Enter your user information



- Click on "Install" to start the installation



- Select "Finish" to end the setup process



After the Installation

After a successful installation the TwinCAT System could be set to CONFIG Mode. The Supplement-Product "TwinCAT Database Server" is now ready to use. With the help of the XML Configurator, it is possible to configurate the database connections and ADS devices for the TwinCAT Database Server. It is shown here [▶ 15].

5 Database Server functionality

The Database Server provides different communication ways and is configured by a XML configuration file.

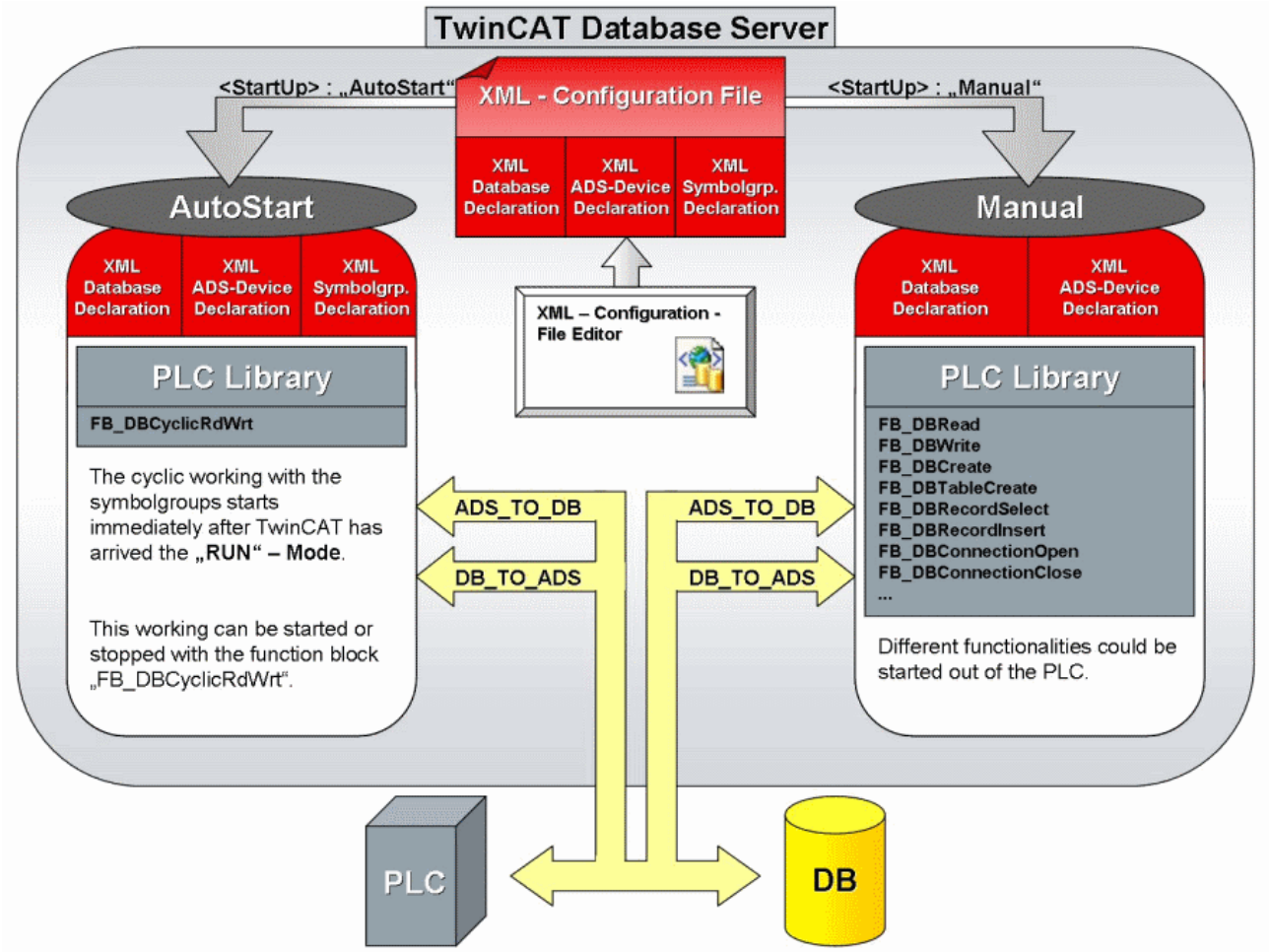
- Communication flow in two directions
  - **ADS_TO_DB**
    Cyclic checking of a TwinCAT ADS device (e.g., TwinCAT PLC) and writing of these data to a "database"
  - **DB_TO_ADS**
    Cyclic checking of a "database" (e.g., SQL database) and writing of these data to a TwinCAT-ADS device (e.g., a PLC) via ADS
- Configuration of the "TwinCAT Database Server" is based on an XML file. This configuration file describes the required "databases", ADS devices (e.g., PLC runtime systems), and variables.
  Two storage methods:
  - **"Double"** The compatible variable data types are: BOOL, LREAL, REAL, INT, DINT, USINT, BYTE, UDINT, DWORD, UINT, WORD, SINT
  - **"Bytes"** Compatible to all variable data types especially for strings and data structures

Two different function modes:

- **AutoStart:**
  Starts the cyclic checking of the PLC values from an ADS device (ADS_TO_DB) or the cyclic checking of a database (DB_TO_ADS) automatically, as soon as the TwinCAT System is an "RUN" - mode. The checking PLC program should be running as a bootproject.
- **Manual:**
  Functions like logging in a database or reading from a database could be started out of the PLC with function blocks.

The following diagram illustrates the server functionality.

The central unit of the TwinCAT 3 Database Server is the XML-Configuration file editor from which all needed options and configurations are made. The created Configuration could be used in two different modes (AutoStart/Manual). The TwinCAT 3 Database Server is the connective link between the PLC and the database.



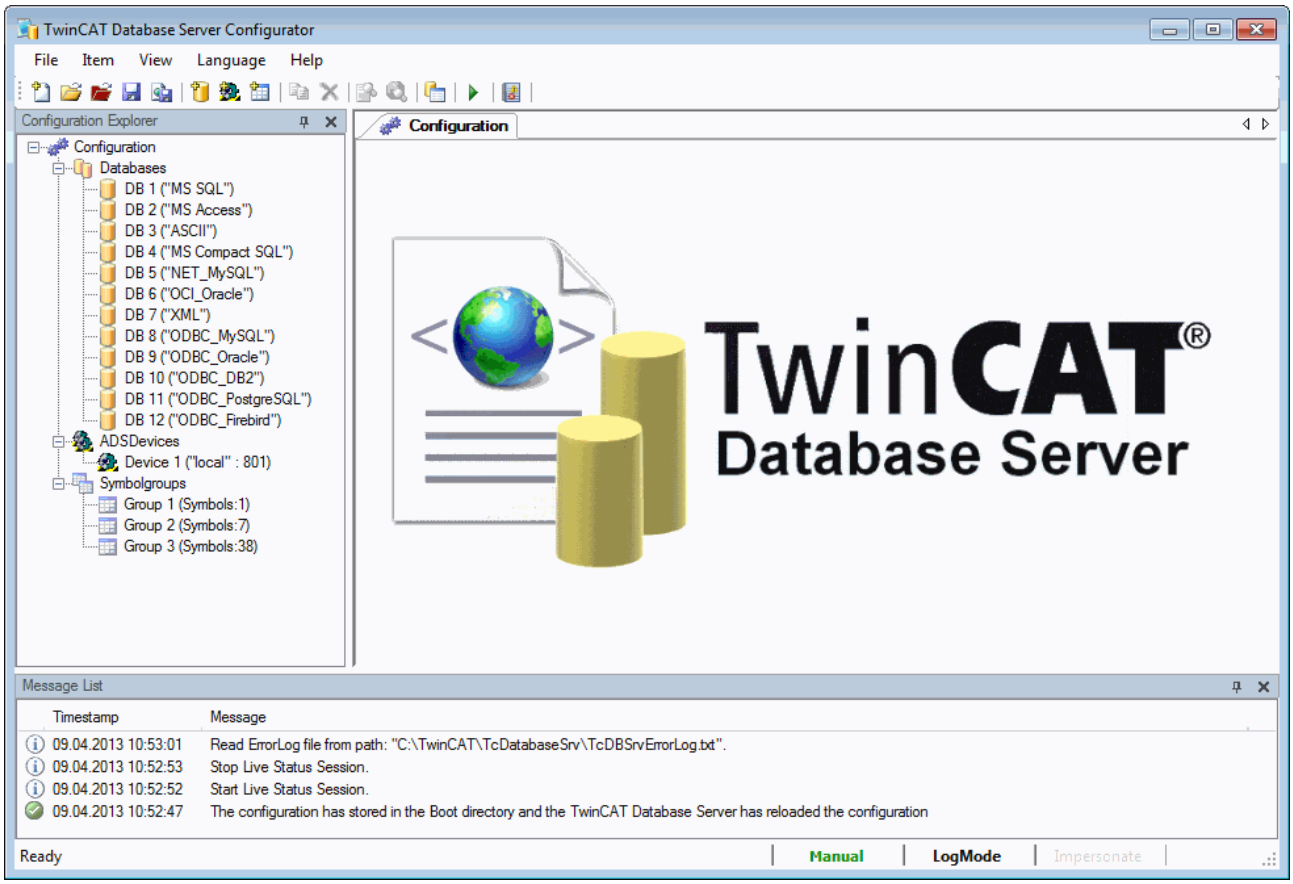
Important!!

Errors can occur because of byte alignment when logging structures from ADS devices with ARM processors.
Errors can occur when logging structures from the BC9000 with floating point variables.

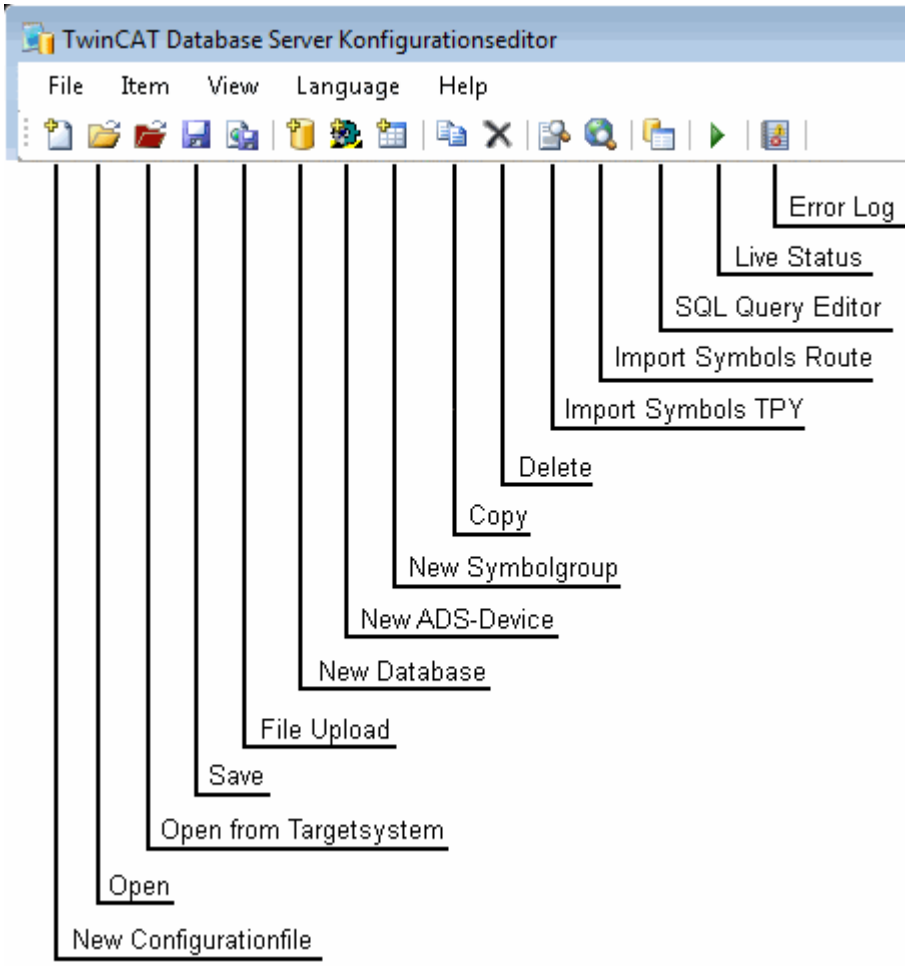
6 Configuration

6.1 XML - configuration file editor

The TwinCAT Database Server is configured via 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, or existing configuration files can be read in and edited.



The Menu Bar



	Description
New configuration file	Creates an empty configuration file with default settings.
Open	Opens an existing XML configuration file.
Save	Saves all the changes that have been made, creating a configuration file with the name: "CurrentConfigDataBase.xml".
File Upload	Uploading of the configuration file to the Database Server, saves the configuration file at the specified "Boot"-directory and activates the configuration.
New database	Creates a new database configuration entry.
New ADS device	Creates a new ADS device configuration entry.
New symbol group	Creates a new symbol group configuration entry.
Delete	Deletes the selected configuration entry. This can be a database, an ADS device or a symbol group.
Insert symbols TPY	Imports symbols/variables from a TPY file into the selected symbol group.
Insert symbols ROUTE	Imports symbols/variables from a Runtime system of a specified route.
SQL Query Editor	Editor for generate SQL commands easily. Further information here [▶ 26].
Live Status	Shows the current state of the TF6420 Database Server and start or stop the cyclic read/write function.
Error Log	Shows the logged errors of the error logfile "TcDBSrvErrorLog.txt"

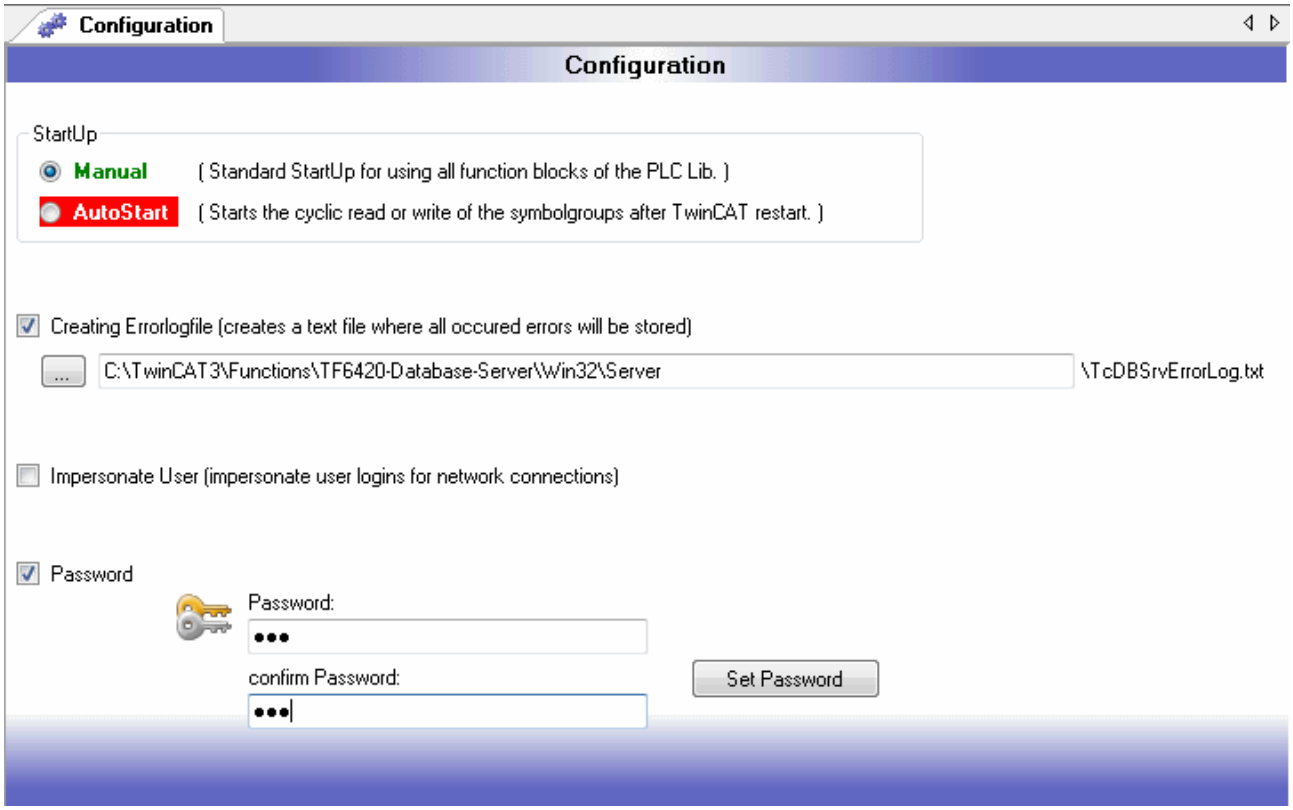
Option dialog

This dialog can be used to set options for the TwinCAT Database Server:

- **StartUp option:**
  two options are available.

- "Manual" => TwinCAT Database Server is active and waiting for function calls from the PLC, with the aid of function blocks from TcDatabase.lib.
- "AutoStart" => TwinCAT Database Server is active and starts logging of the set symbol groups as soon as the TwinCAT system is in "RUN" mode.
  PLC programs from which variable values are to be logged must run as boot projects!

- **ErrorLog option:**
  enable the error log mechanism. Any errors that occur are logged to a text file. The logged errors can then be used for error analysis.
  Additionally, the storage path of this text file can be specified.
- **Impersonate option:**
  the Impersonate option must be set for network access to file-based databases such as Access databases or SQL Compact databases so that the TwinCAT Database Server can connect to this network drive. For further information see Impersonate [▶ 53]. **This feature is currently not supported in Windows CE.**

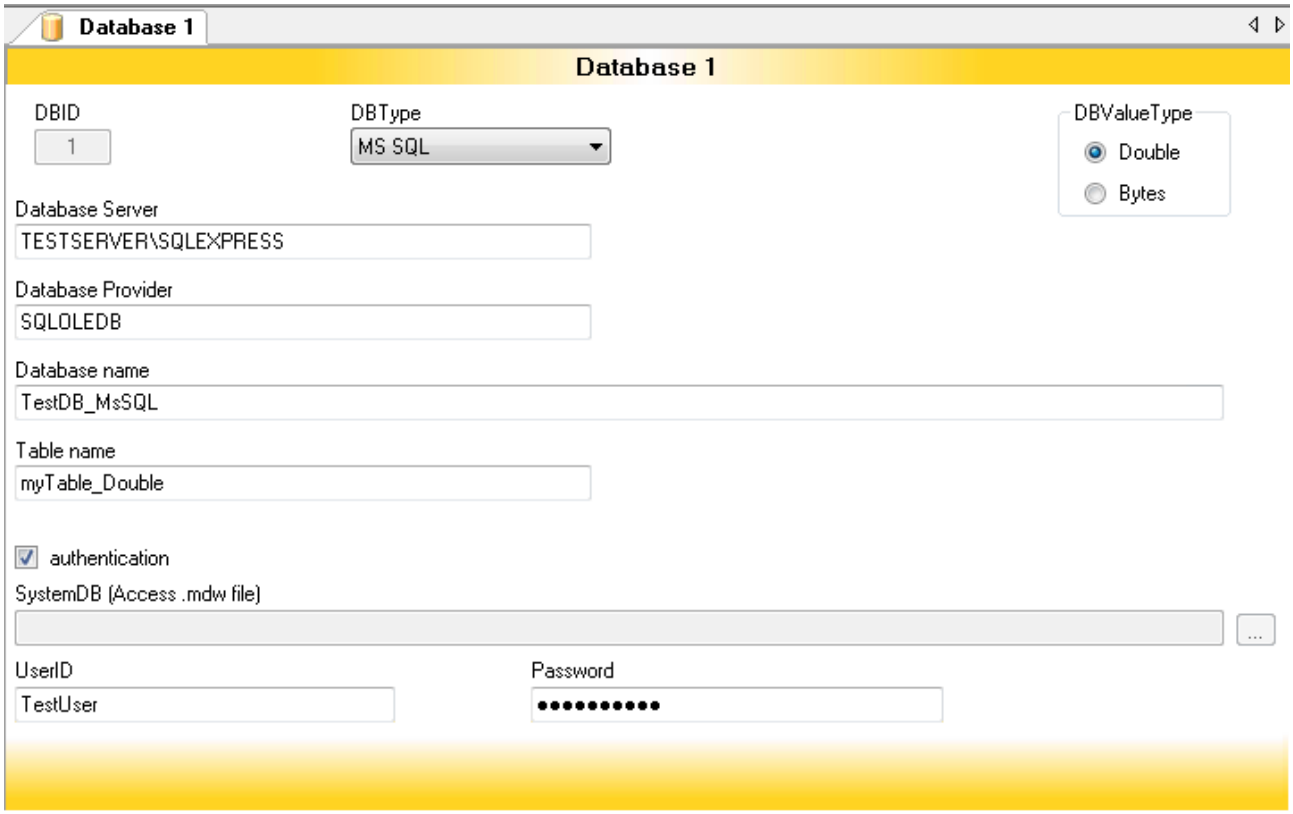


Database Configuration Dialog

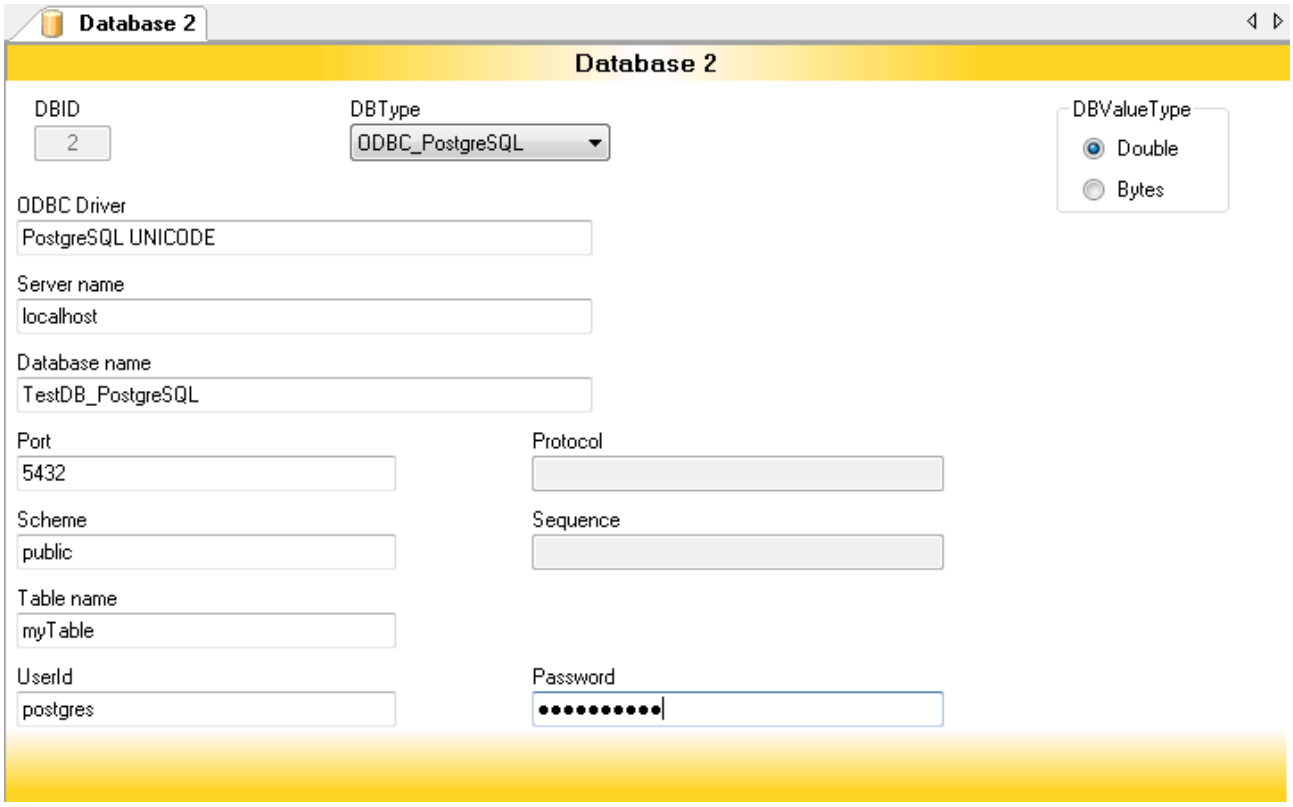
All the information required for database communication can be entered in this dialog. Only fields for the necessary entries are enabled.

The DBValueType indicates the data type of the "Value" column. PLC structures and strings cannot be logged if the data type is "double". Structures and strings can be logged if the data type is "bytes".
You can choose if the database communication needs an authentication or not. If you want to add authentication information, the fields **DBSystemDB**, **DBUserId** and **DBPassword** will be enable.
The field **DBSystemDB** is needed for Access databases only. In this field you must write the path of the MDW-file. In this file all information like usernames and passwords are saved.
DBUserId is the username and **DBPassword** is the necessary password.

configuration of all database types: database-configuration [▶ 35]

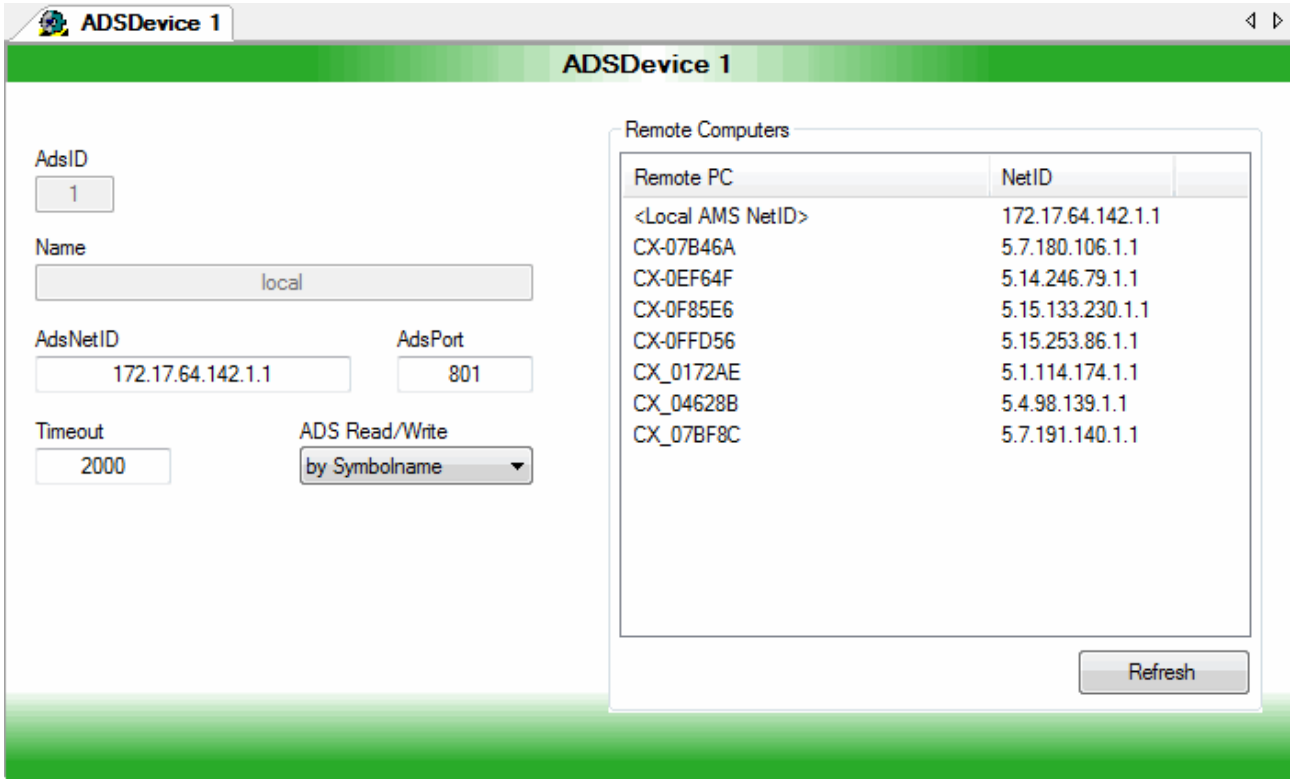


If you choose ODBC-database types the input mask will be changed. Further information for the ODBC-connection like Port, Protocol, Driver, Scheme and Sequence must be insert.



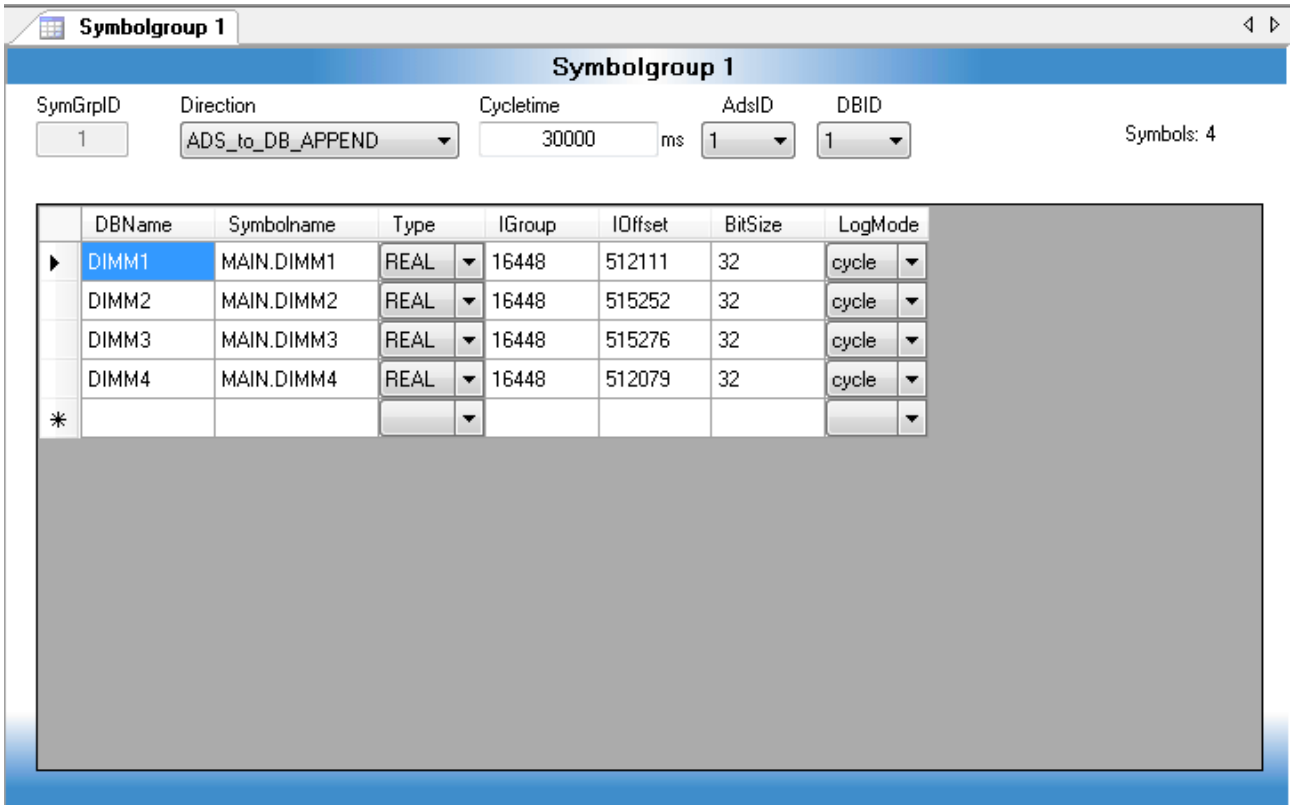
ADS Device Configuration Dialog

All the information required for communication with the ADS devices can be entered in this dialog. For easier input all declared RemotePCs of the TwinCAT System will be listed and could be selected. Of course, it is possible to insert other NetIds, which are not declared as RemotePCs.



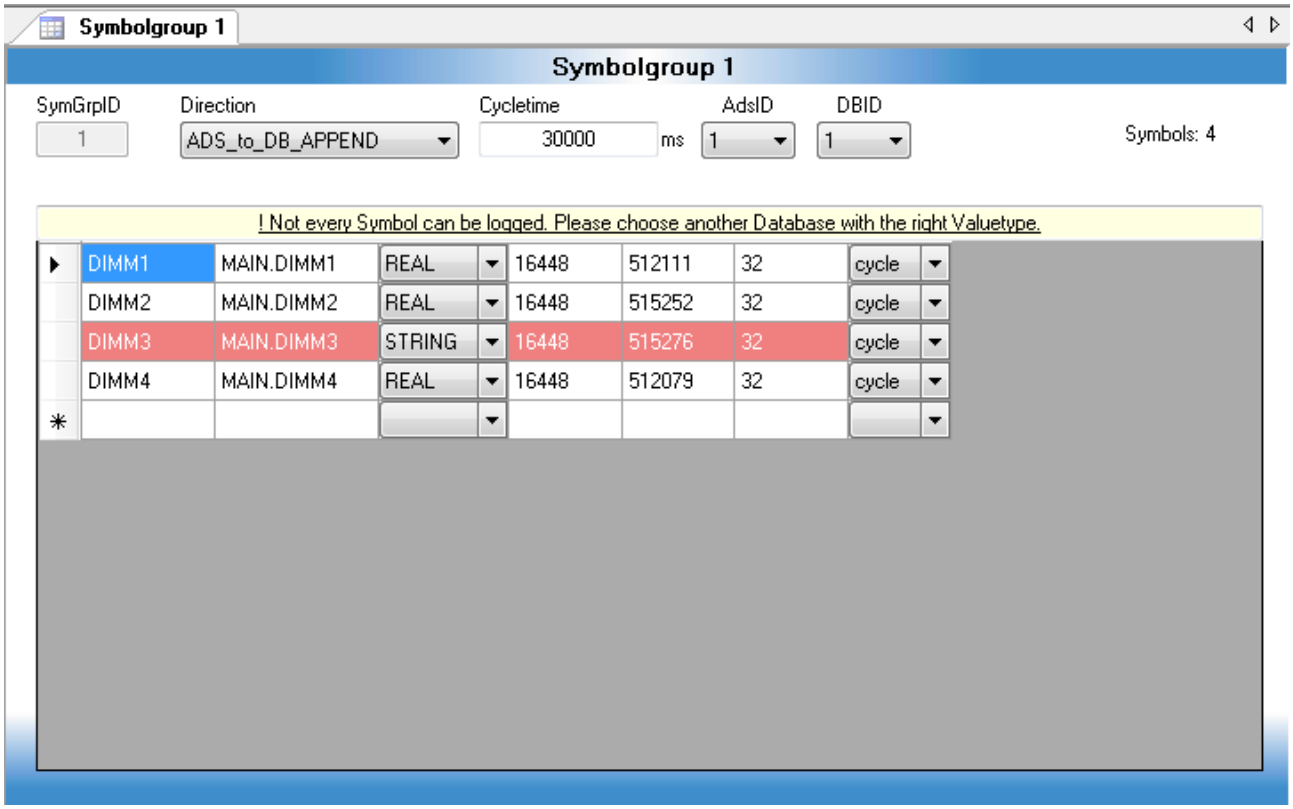
Symbol Groups Configuration Dialog

Symbol groups can be arranged in this dialog and assigned to ADS devices or databases. In addition, the writing direction of the symbol group is specified. There are various setting options see Write direction mode [▶ 32].

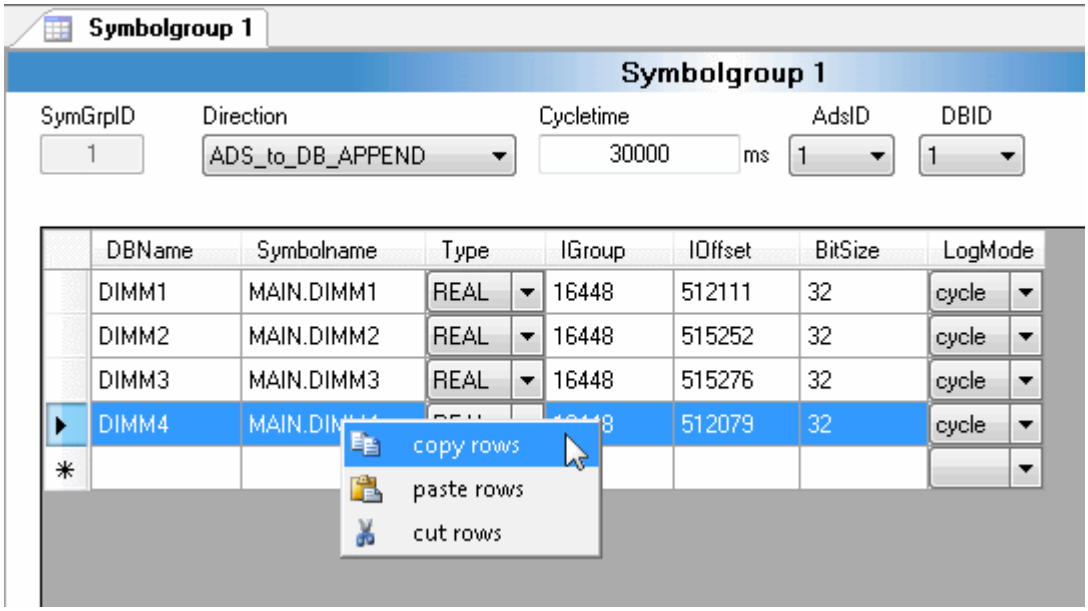


This dialog also contains a symbol counter. It returns the number of symbols declared in the individual groups. If more than 500 symbols are declared, the display turns red, as no more than 500 symbols/ variables may be declared for each symbol group.

If, due to the database settings, data types are not supported, these symbols are highlighted in red and a warning message is displayed in the upper region.



Of course, symbol rows could be copy and paste. You only must select a row and click the right mouse button. A context menu open and you can choose the desired function.

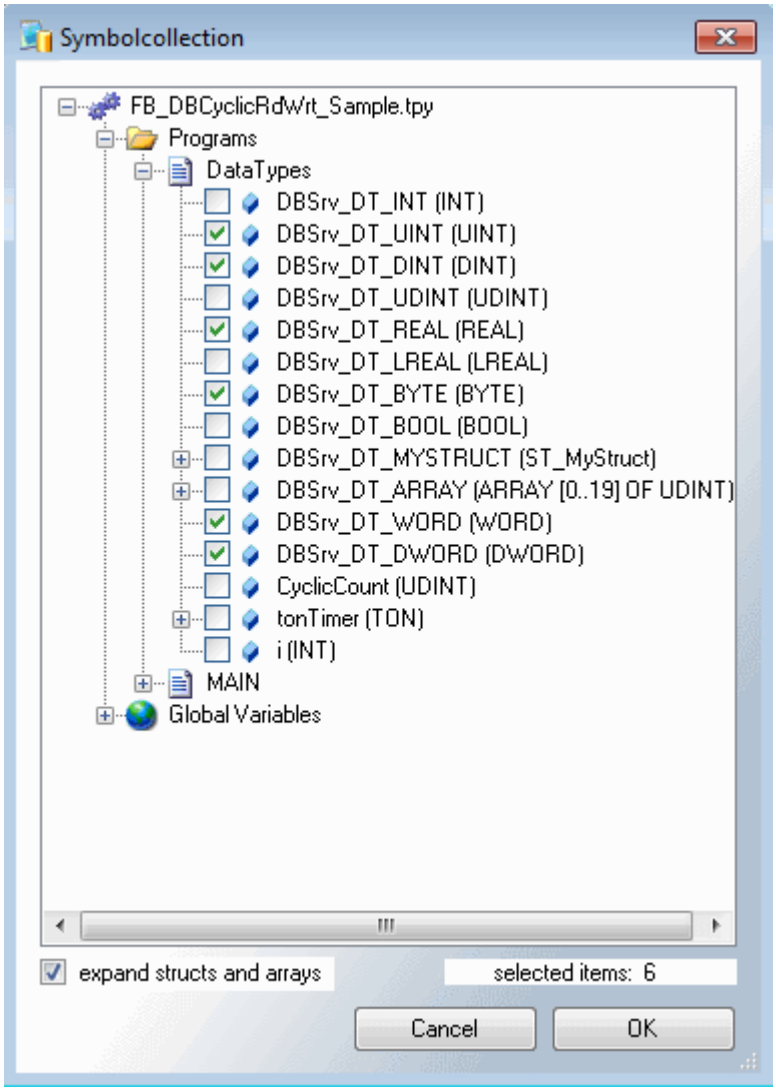


Import of symbols

It gives two possibilities to import symbols into the TwinCAT Database XML configurator. One way is to read out the TPY file. The other way is directly from the target device with the Target Browser.

Symbols from TPY

All symbols of this project will be listed very clearly in the following dialog. Also, you can import each element of an array or of a structure into the symbol group. So, you can log arrays and structs which consist of numeric data types if the selected DBValueTyp is "Double".

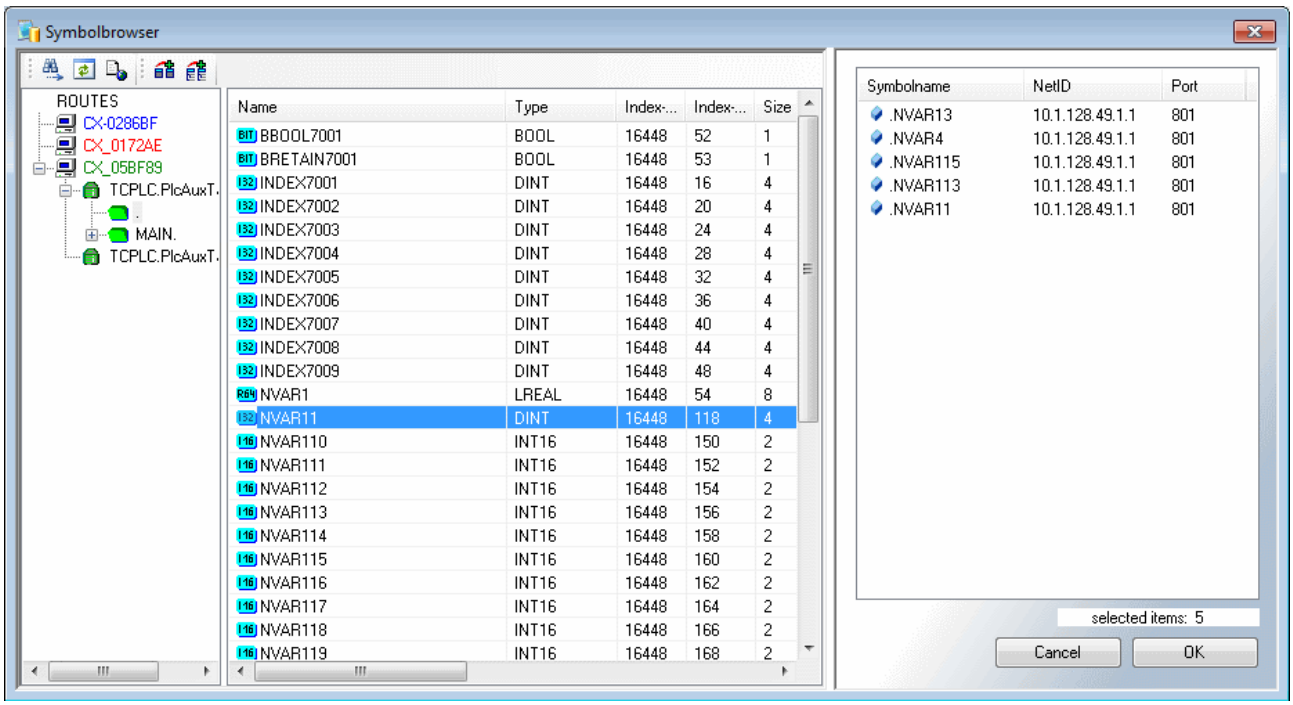


Directly from the target with the Target Browser

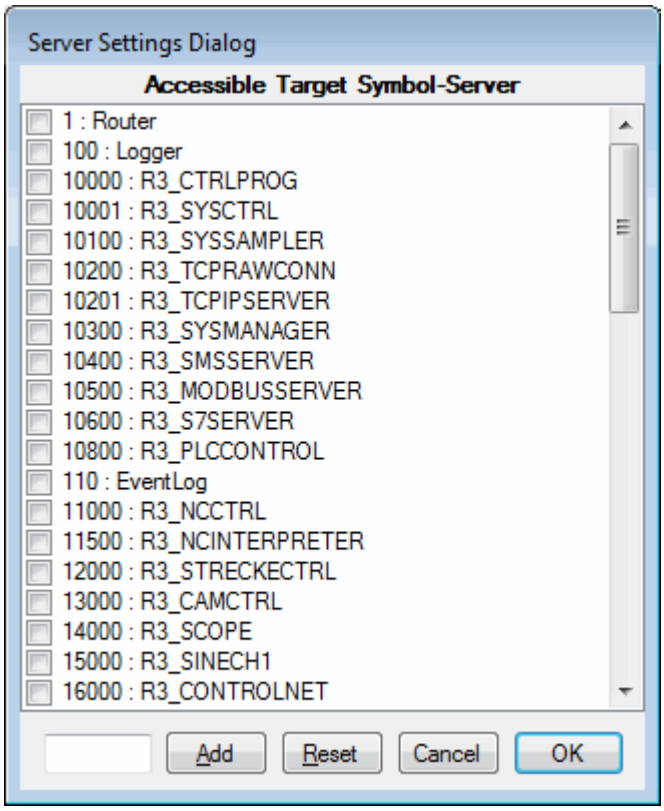
The Target Browse is used to add channels by a known Symbol to the configuration. The Target Browser is separated in three parts. The left one shows a tree view with the root named ROUTES. Beneath all TwinCAT System Manager known targets are listed. The color of the nodes explains the system state: Red= not Connected (Stop-Mode), Blue= Config Mode, Green= Run Mode.

The second part contains a list view showing the details of the selected node in the tree view.

In the third part is a list view which show all choose symbols.



It is possible to add new ADS ports in the Target Browser. So, it is realizable to log values directly from EtherCAT terminals, if the ADS port is enabled in the TwinCAT System Manager under EtherCAT Device Image.

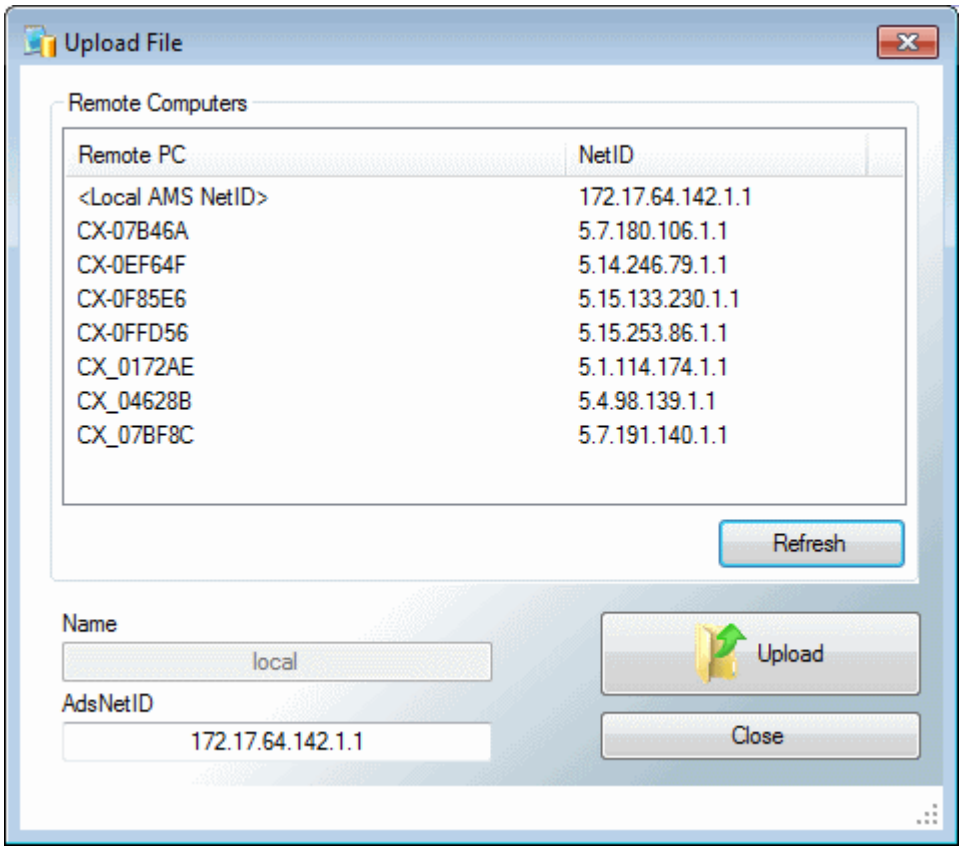


Uploading the XML configuration file to the TwinCAT Database Server

The created XML configuration file must be copied into the corresponding "TwinCAT\Boot" directory for activation and uploaded to the TwinCAT Database Server by a restart of the TwinCAT system, or by the function block FB_DBReloadConfig.

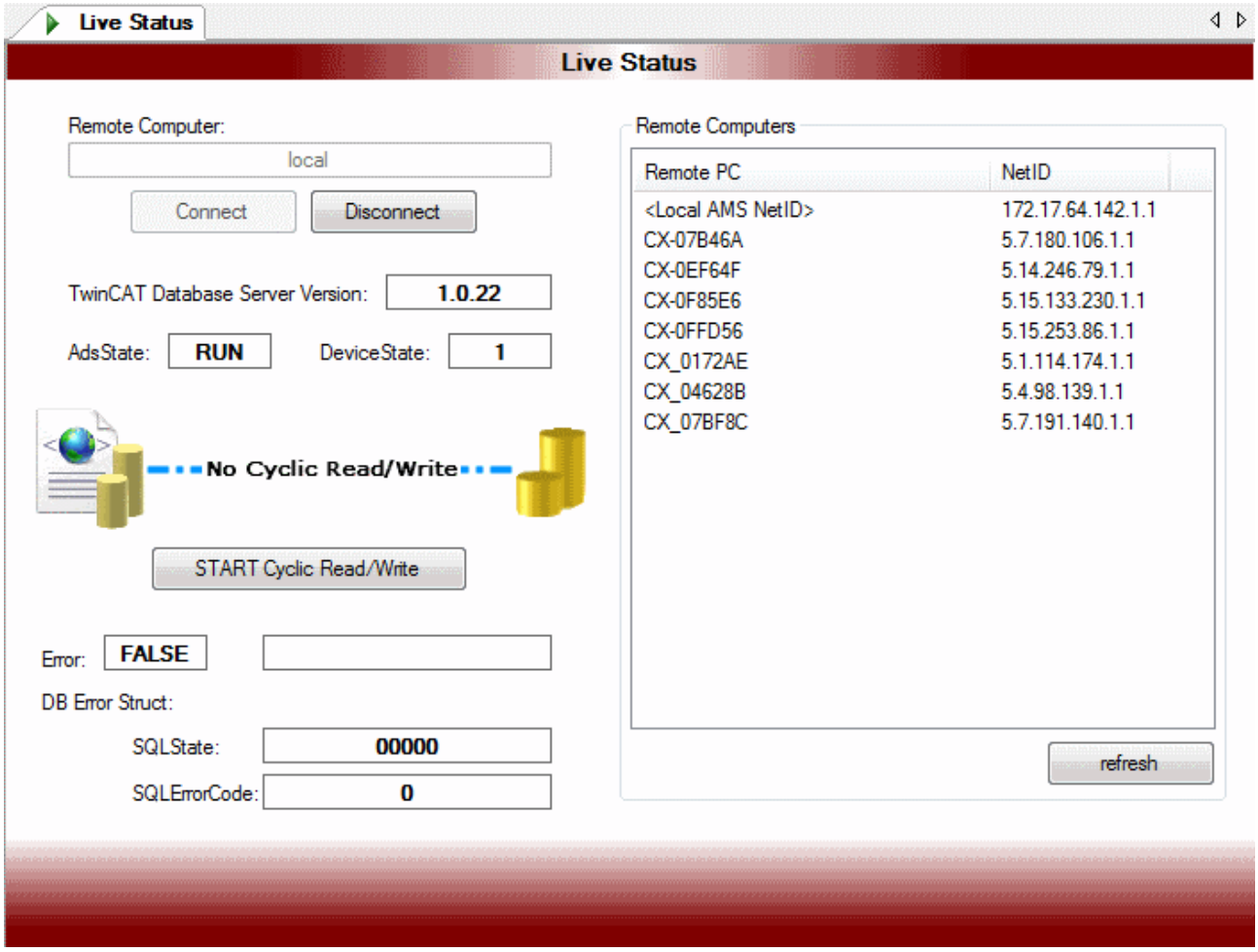
Another option is to activate the generated file using the "Upload" dialog. It is only necessary to select the desired ADS device and start the process with the Upload button. The XML configuration file is then transmitted to the TwinCAT Database Server, stored in the boot directory and read.

To use the service, TwinCAT must be installed and in RUN mode on the host PC running the XML configuration file editor. Furthermore, the TwinCAT Database Server must be started on the corresponding ADS device.



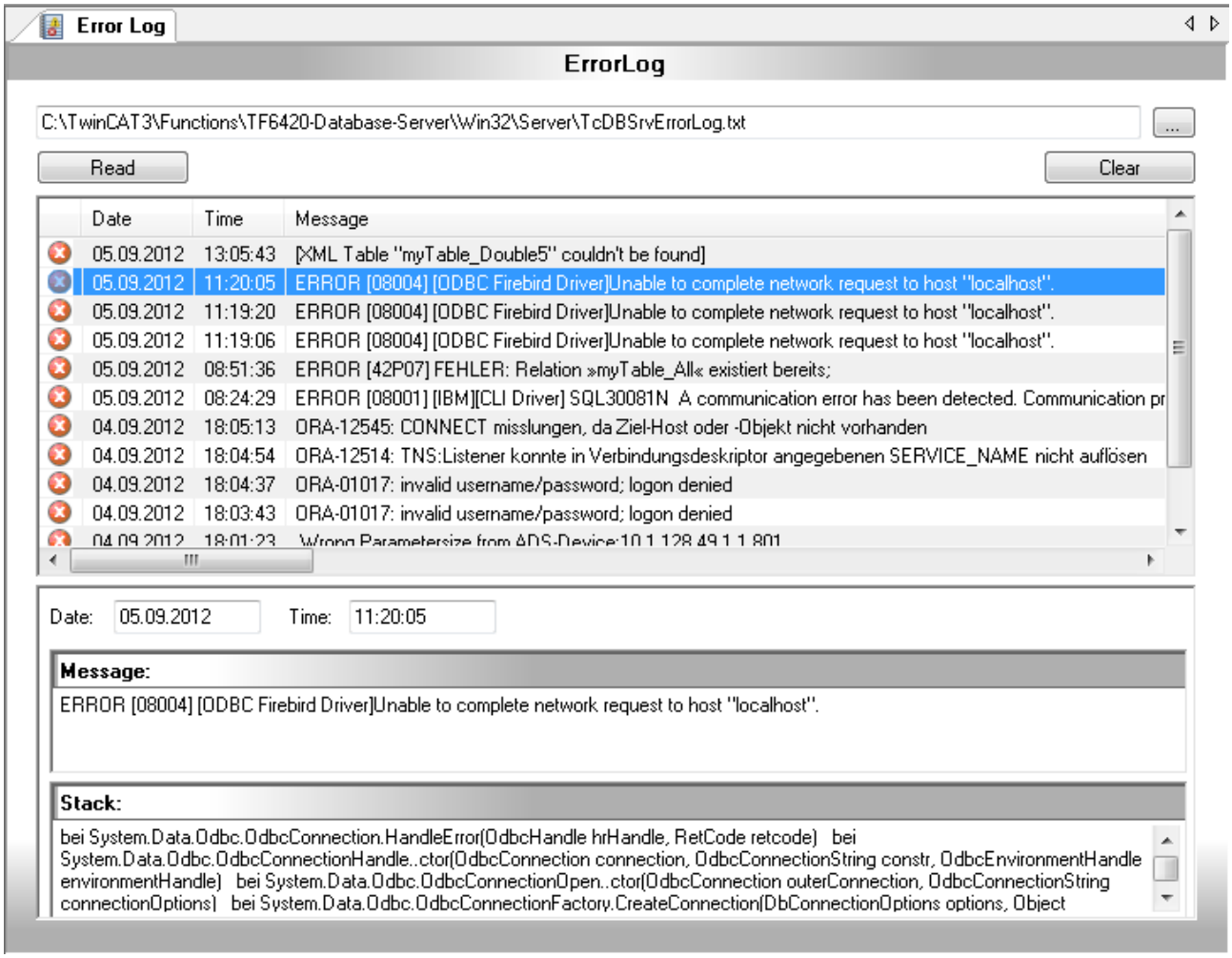
Live Status of the TF6420 Database Server

At this dialog the current state of the Database Server will be shown. It is possible to check the status of the Database Server from a remote computer. It is also possible to start or stop the cyclic read/write functionality of the Database Server. If error occurs during the cyclic read/write process, the sqlstate and the errorcode of the occured error will be shown too.



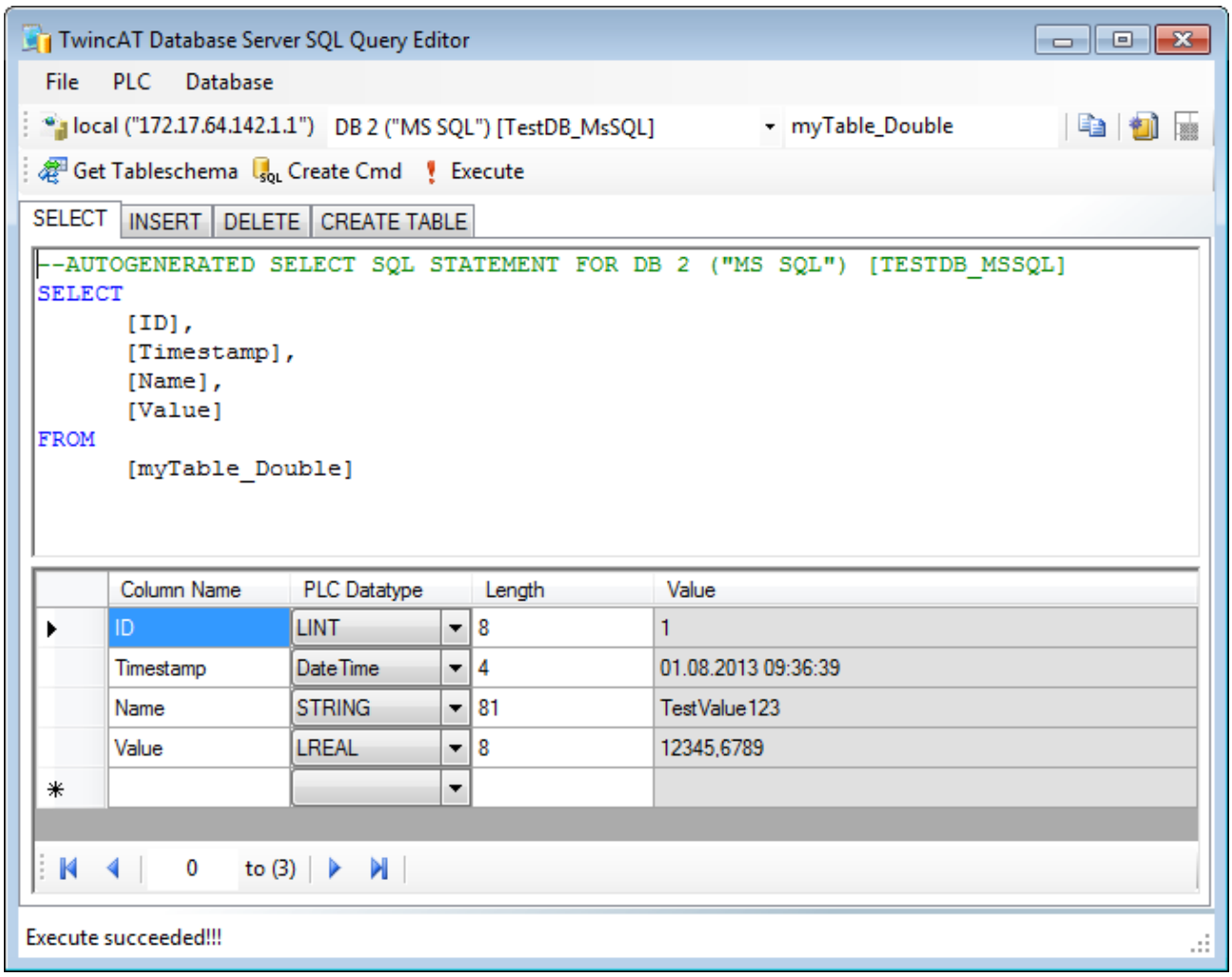
Error Log (TcDBSrvErrorLog.txt)

At this dialog all items of the error log file "TcDBSrvErrorLog.txt" will be shown. It is also possible to clear the error log file.



SQL Query Editor

The SQL Query Editor helps to generate SQL commands and test the configured database connections. Further information here [▶ 26]



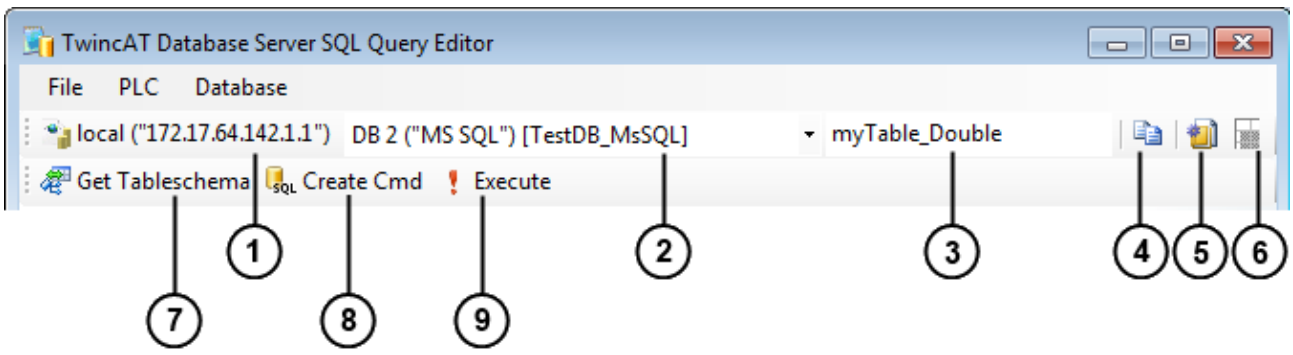
6.2 SQL query editor

The TwinCAT Database server combine two different worlds. On the one hand the know-how of automation engineers and on the other hand the IT database administrators.

To work more efficent the SQL Query Editor would be developed.

With the editor you can generate SQL commands and test them with standard PLC function blocks of the TwinCAT Database Server. Additional it is possible to export the generated commands or structs to TwinCAT. It is also possible to select, insert, delete datasets, or create tables. The generated SQL commands based of the database specific syntax. The different functionalities of the editor will be sent to the configured database connections.

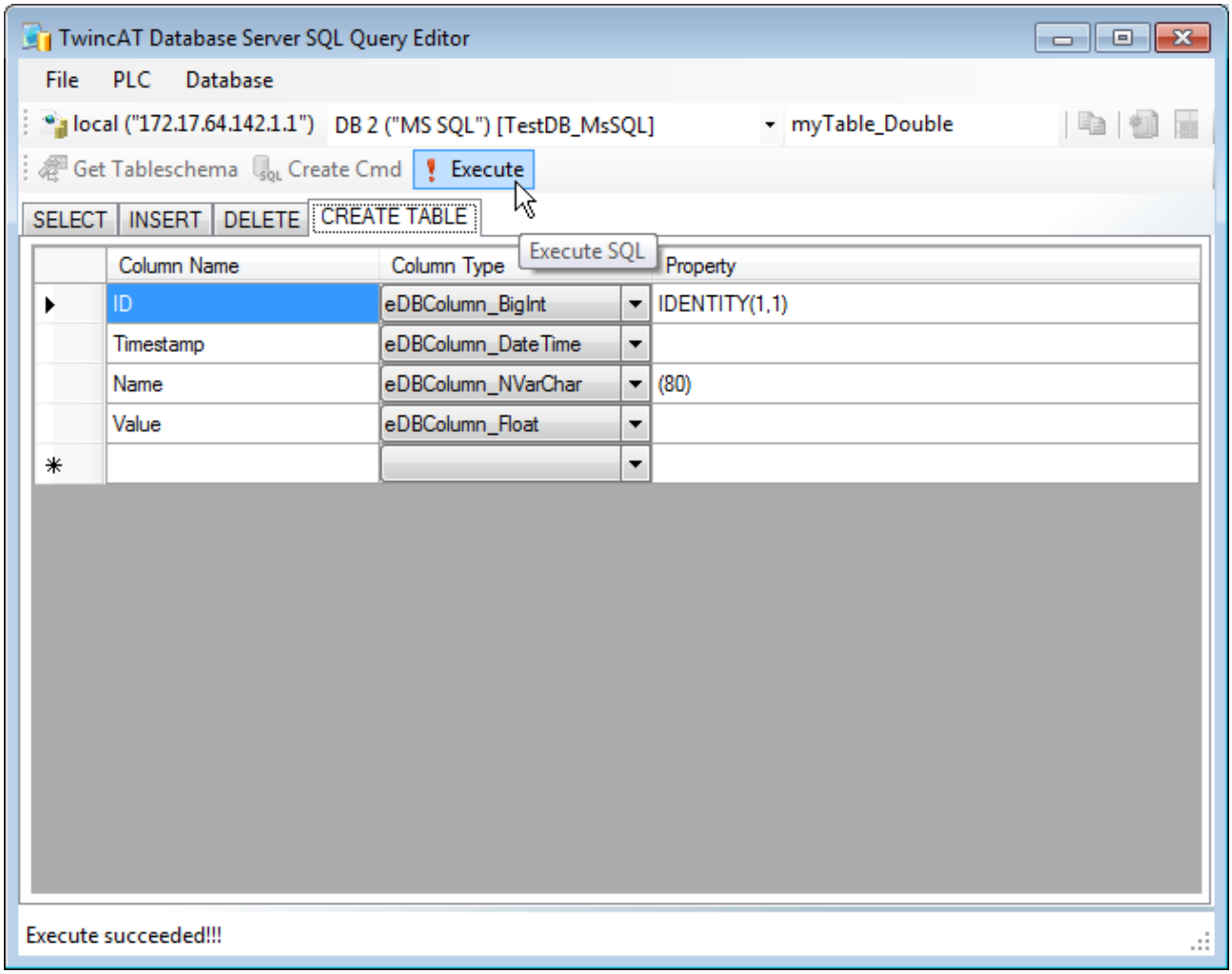
The Menu Bar



		Description
1	**Target TwinCAT Database Server**	Choose the Target TwinCAT Database Server for communication.
2	**Database**	List of the configurated database connections
3	**Table**	Tablenname for SQL command generating
4	**Copy for PLC**	Copy the created SQL command with correct PLC syntax into the clipboard
5	**Export TC2**	Create an export file for TwinCAT 2 for the SELECT PLC struct
6	**Export TC3**	Create an export file for TwinCAT 3 for the SELECT PLC struct
7	**Get Tableschema**	Reads the table structure of the given table
8	**Create Cmd**	Create SQL commands based of the different database syntax
9	**Execute**	Executes the SQL command

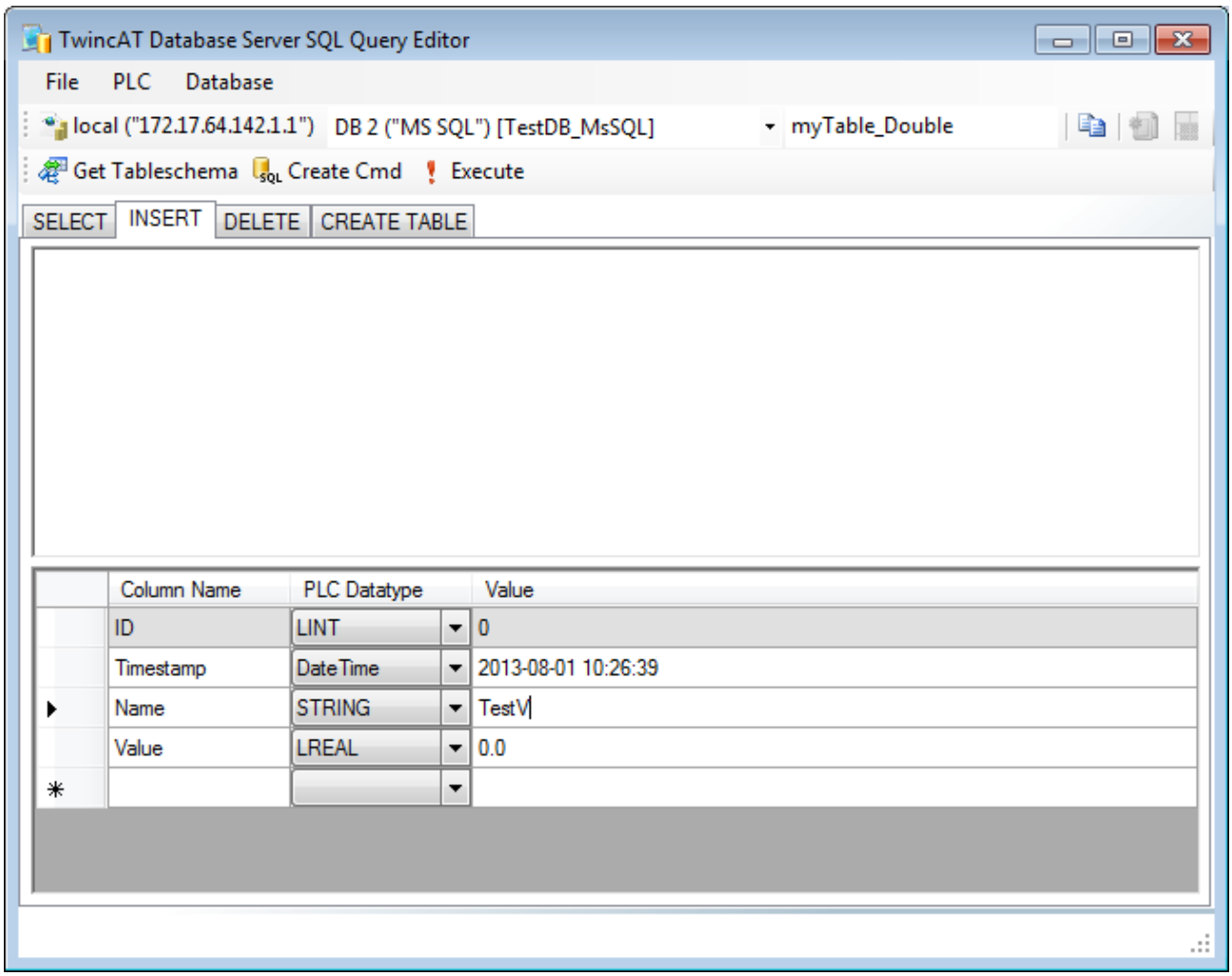
SQL "CREATE TABLE"

With the tab "CREATE TABLE" you can create arbitrary tables at the selected database. This operation will be executed internally with the function block FB_DBTableCreate. With the table text field (3) it is possible to indicate the name of the new table.

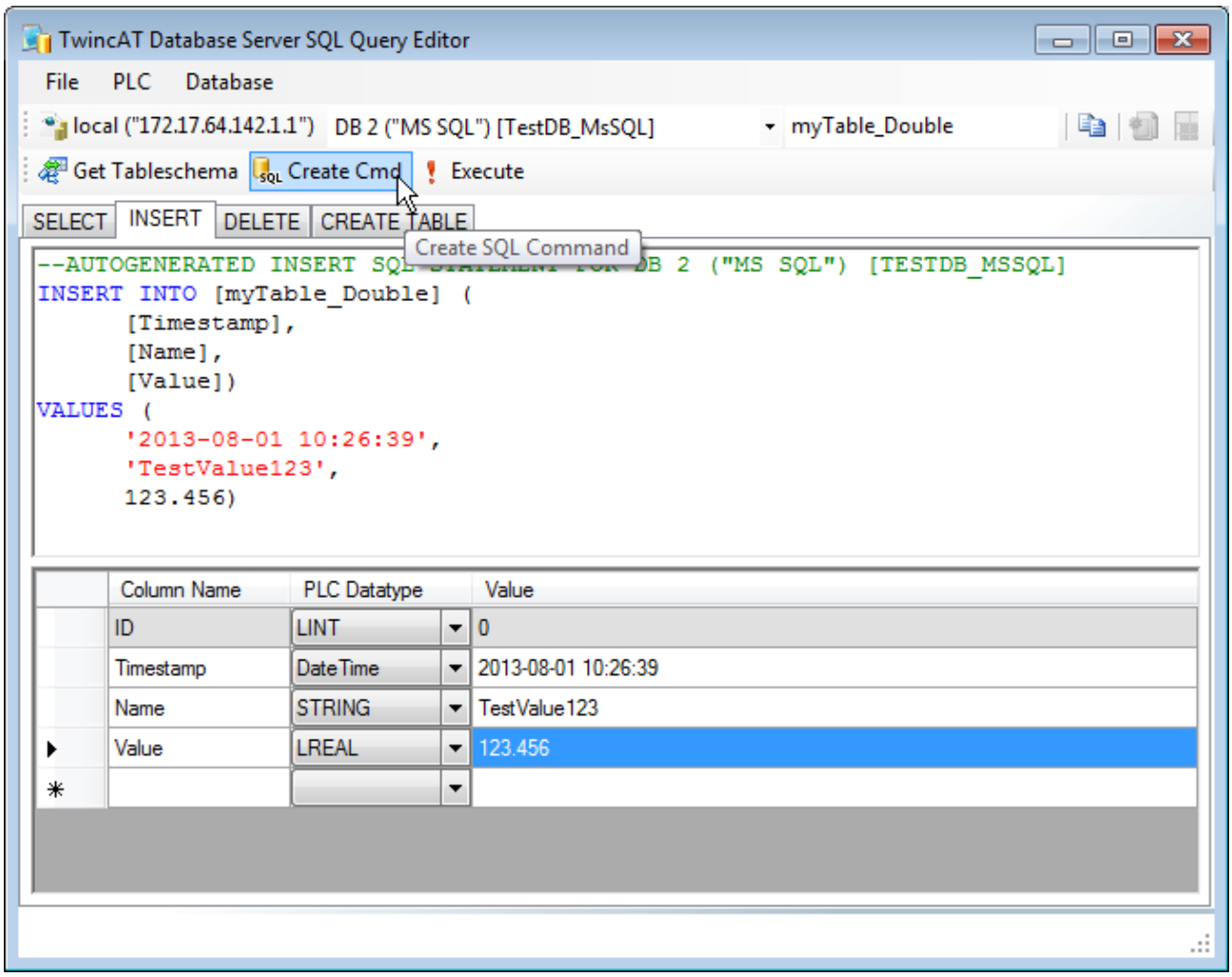


SQL "INSERT" commands

With the tab "INSERT", INSERT SQL commands could be created of a fast and easy way. You can configure the individual columns of the table by hand, or you read the whole table schema with the help of the button "Get Tableschema".

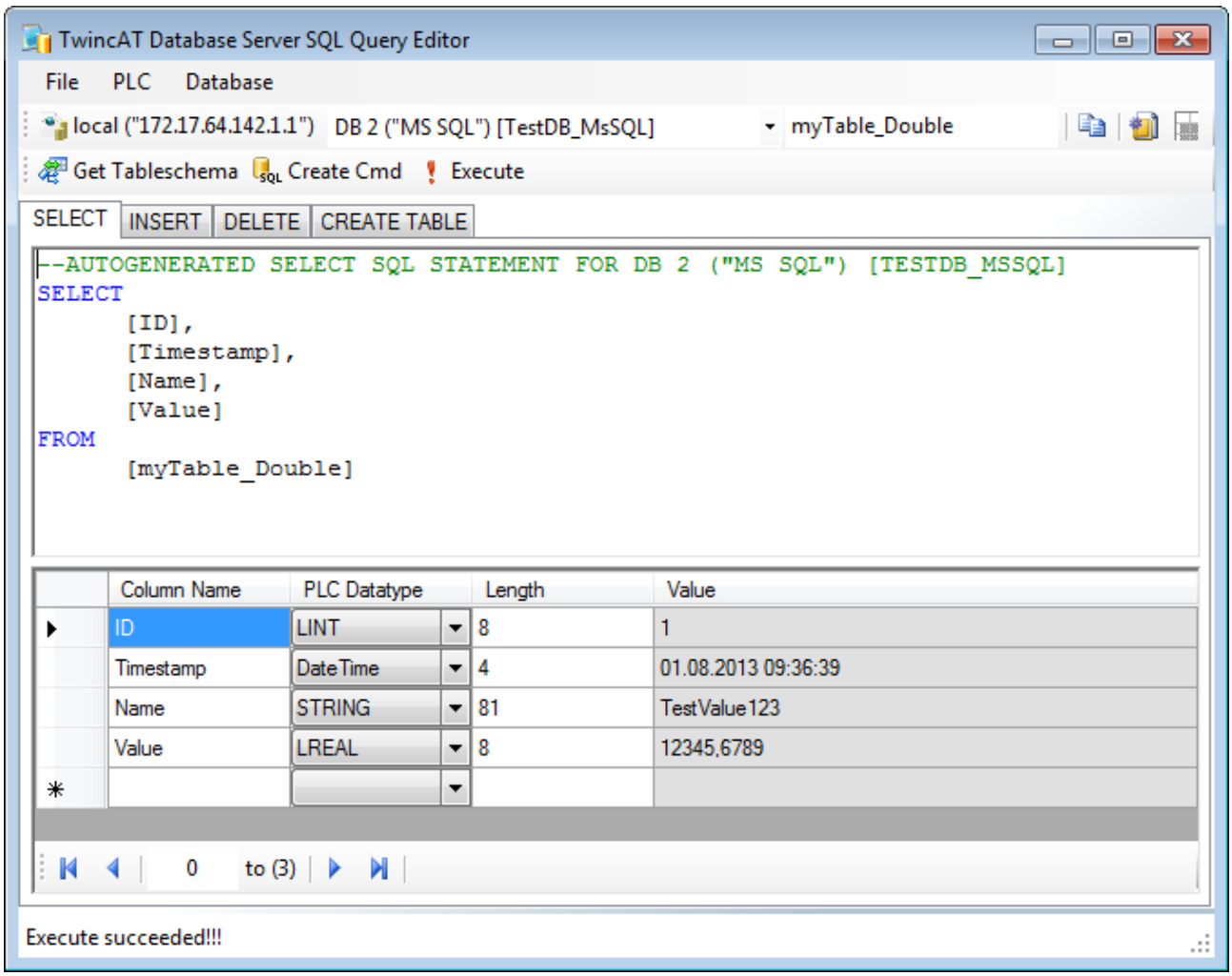


If all column values are insert, and the button "Create Cmd" would be clicked, the INSERT SQL command will be created with the right database syntax. After that, you only must press the "Execute" button to send the command to the database. This operation will be executed internally with the function block FB_DBRecordInsert_EX



SQL "SELECT" commands

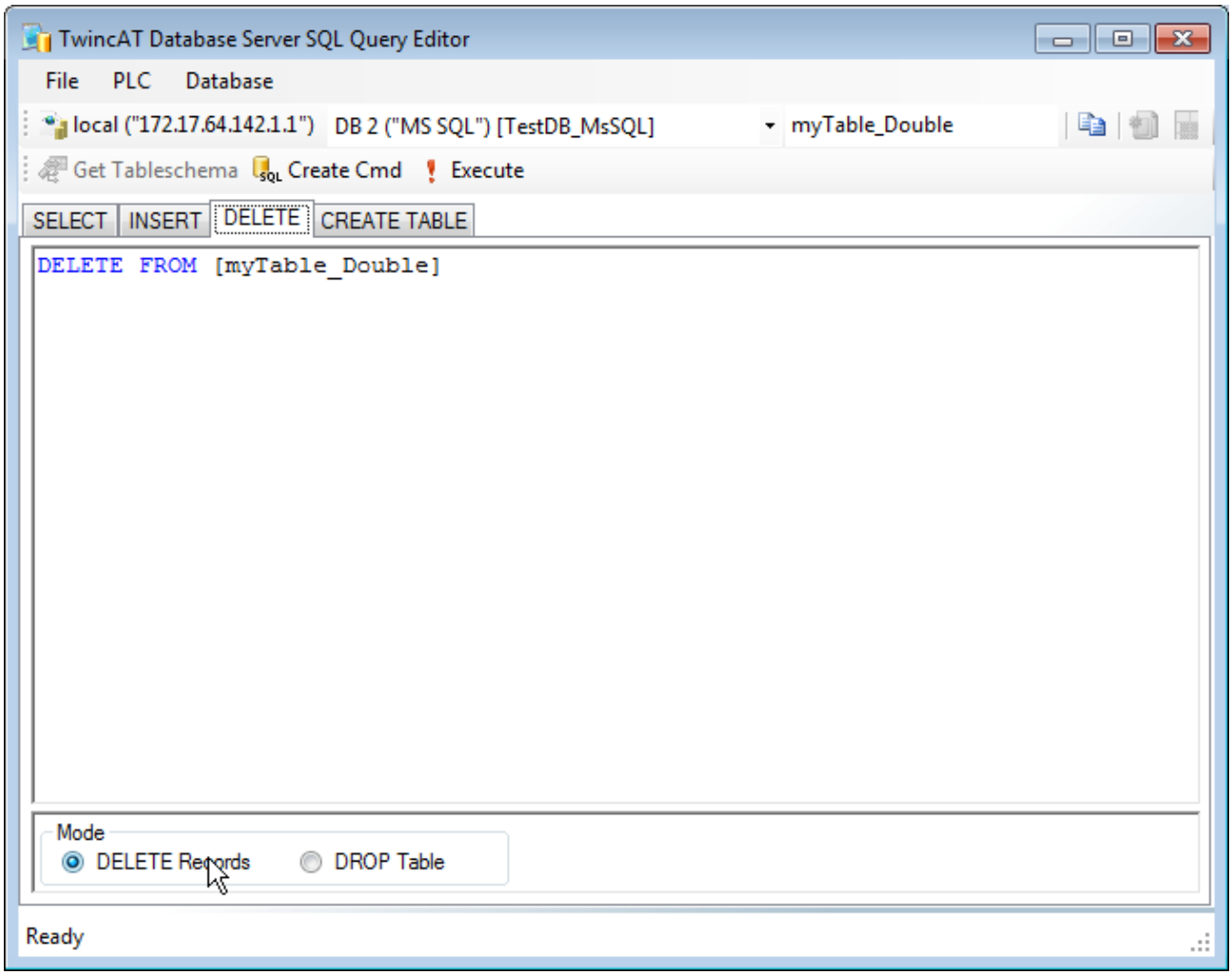
With the tab "SELECT" datasets of a table could be read. Therefore, you must declare at the bottom of the tab the struct with standard PLC data types in which the data will be saved. You can do that by your own or you use the button "Get Tableschema". The created struct can be export to the TwinCAT PLC program. This operation will be executed internally with the function block FB_DBRecordArraySelect.



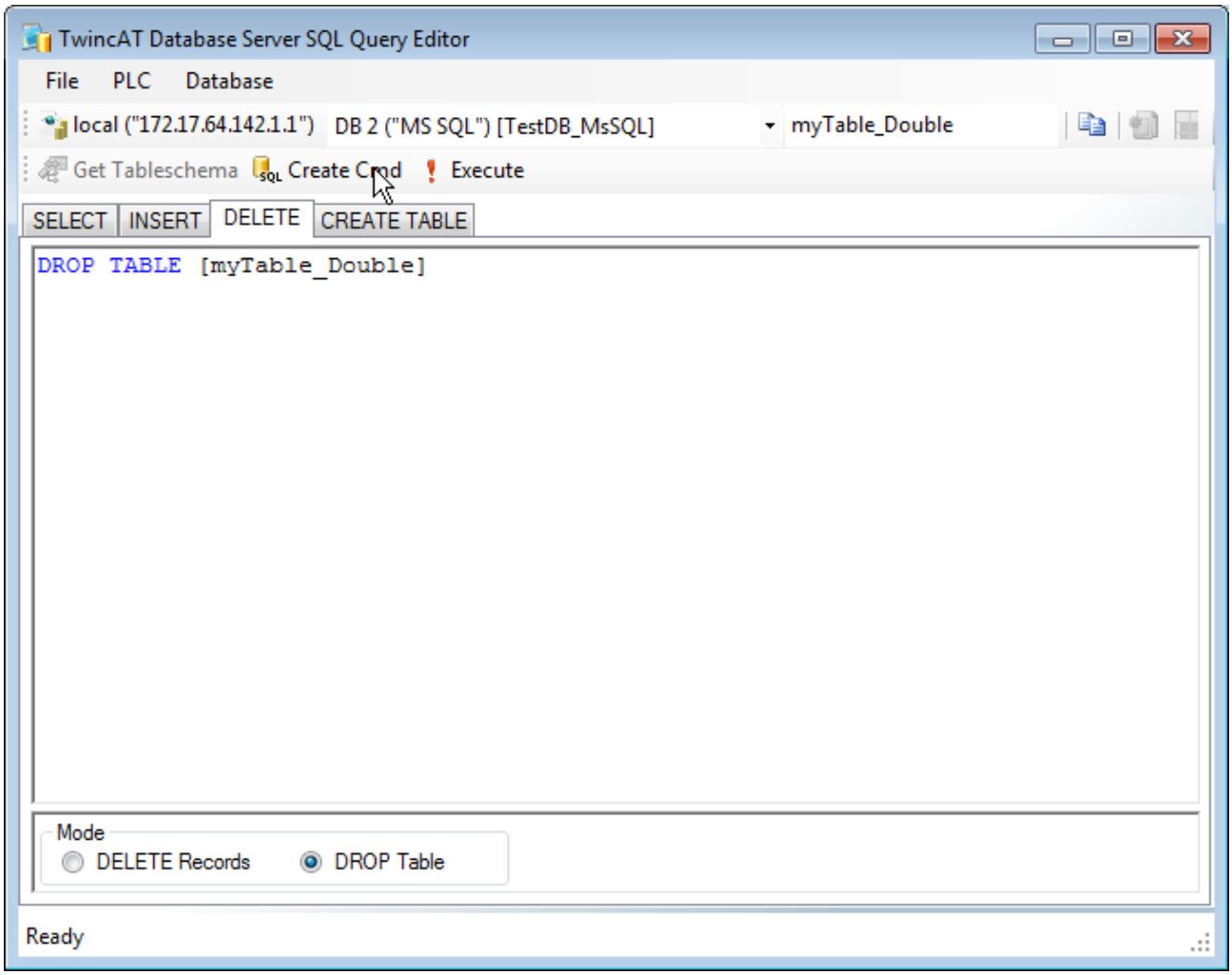
SQL "DELETE" commands

Of course, the SQL Query Editor contains a possibility to delete datasets or complete tables. For that purpose, the SQL Query Editor contains the tab "DELETE". Here it is possible to generate DELETE SQL commands. Internally the commands will be executed with the function block FB_DBRecordDelete.

DELETE records



DELETE tables



6.3 Write Direction Mode

The TwinCAT Database Server provide four different wirte direction modes.

DB_TO_ADS

With this write mode it is possible to read cyclic values out of a database and write them in variables in the PLC.

ADS_TO_DB_APPEND

With this write mode it is possible to write cyclic values from the PLC into a database. Each cycle a new record is created and added to the end of the table / file.

ADS_TO_DB_UPDATE

With this write mode values will be cyclically read out of the PLC. These values will be compared with the records in the database. If the values differ the specified record will be updated with the new value.

ADS_TO_DB_RINGBUFFER

With this write mode you can limit the count or the age of datasets on database tables.
This write mode is available for the cyclic logging with symbol groups and for logging with the function block FB_DBWrite.
Every databasetype can be used with this write mode. Also logging in ASCII-files can be influenced with the

RingBufferMode.
"RingBuffer"-Versions:
The RingBuffer works in two different ways:

- "RingBuffer_Time"
- "RingBuffer_Count"

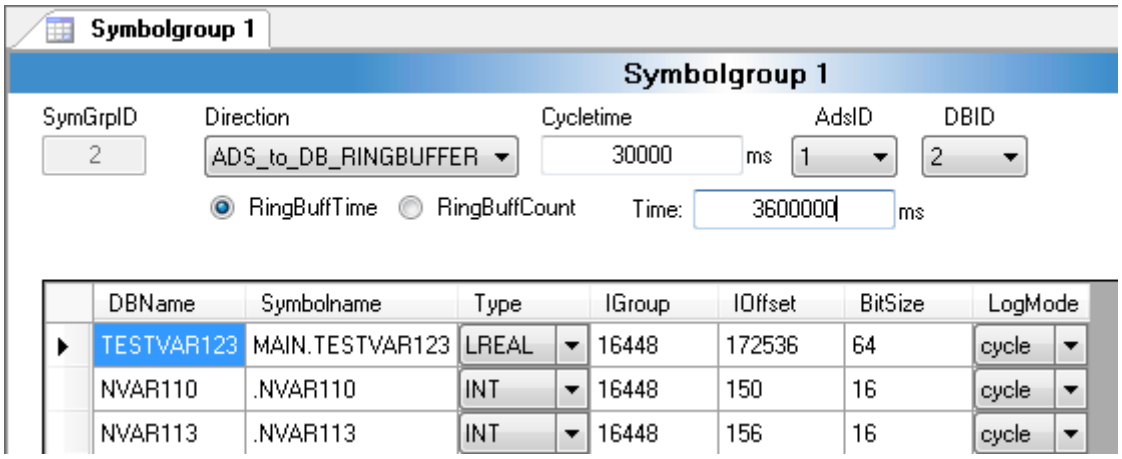
RingBuffer "Time":
In this mode a timestamp can be set, this timestamp defines the age of the datasets. If this age is exceeded, the affected datasets will be deleted.

RingBuffer "Count":
In this mode a maximum count of datasets can be defined. If the maximum count is obtained, the oldes datasets will be deleted to get space for new datasets.

Declaration of the RingBuffer Mode:

Table 1: XML configuration file editor:

RingBuffer_Time



The time is given in milliseconds.

RingBuffer_Count

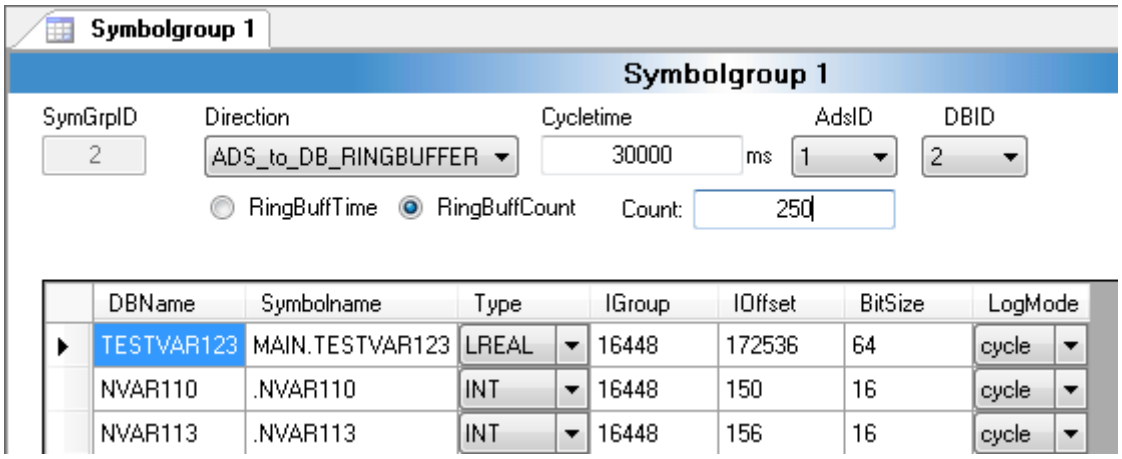


Table 2: FB_DBWrite:

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

6.4 Properties and use of the XML-configuration file

Storage place of the XML-configuration file:

The configuration file has a fixed storage place.

- On CE devices the configuration file is placed at the following folder "\Hard Disk\TwinCAT\Boot" (If you change the XML-configuration file with the XML-configuration file editor, you have to copy this file back to the folder "\Hard Disk\TwinCAT\Boot")
- On PCs the configuration file is placed at the folder "C:\TwinCAT\Boot"

Loading the XML-configuration file:

The configuration file will be read once if the TcDatabaseSrv.exe is starting.
With the Function block **"FB_DBReloadConfig"** you can read the configuration file once again. (This is only possible, if the cyclic read/write isn't started.)
After every restart of the TwinCAT System the XML-configuration file will be reloaded.

Mode 1 (StartUp = "AutoStart")

The XML configuration file loads automatically, when the TcDatabaseSrv.exe starts. Because of the value "AutoStart" at the tag StartUp, the Database Server starts immediately creating the connections to the declared databases und ADS-devices. Also, all variables, which are described in symbol groups, are logged to the specified database resp. written with values of the database. This process will be executed with the declared cycle time. The process will be continued until the Database Server is stopped by the PLC with the function block **"FB_DBCyclicRdWrt"**

Mode 2 (StartUp = "Manual")

The XML-configuration file loads automatically, when the TcDatabaseSrv.exe starts. Because of the value "Manual" at the tag StartUp, no further function will be executed. After this the Database Server is waiting for commands from the PLC.

"FB_DBCyclicRdWrt"

All sections of the configuration file are needed for this function block.
Create the connection to the databases and the ADS-devices which are declared in the XML-configuration file and the cyclic logging with all symbol groups will be started.

All other function blocks only needs the declared databases and ADS-devices of the configuration file. The symbol groups will be ignored. You can see that at the following function block.

"FB_DBWrite"

A connection to the selected database (hDBID) and the selected ADS-device (hAdsID) will be created. After this the variable which is indicated at the function block will be read out from the ADS-device and logged into the database.

6.5 Databases

6.5.1 Declaration of the different database types

Mircosoft SQL Database [▶ 38]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"MS SQL"**. →PLC: **eDBType_Sequal_Server**.
-DBServer:	Insert here the Name of the sql-server. e.g. ("TESTSERVER\SQLEXPRESS")
-DBProvider:	**"SQLOLEDB"** or Provider of the SQL Native Clients e.g., "**SQLNCLI10**".
-DBName	DBName contains the name of the database.
-DBTable:	DBTable contains the name of the table.

Mircosoft Compact SQL Database [▶ 40]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"MS Compact SQL"**. →PLC: **eDBType_Mobile_Server**.
-DBServer:	This option won't be needed.
-DBProvider:	This option won't be needed.
-DBUrl	DBUrl contains the path to the SDF-File. e.g. ("C: \TwinCAT\TcDatabaseSrv\Samples\TestDB.sdf")
-DBTable:	DBTable contains the name of the table.

Microsoft Access Database [▶ 39]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"MS Access"**. →PLC: **eDBType_Access**.
-DBServer:	This option won't be needed.
-DBProvider:	For Access 2000 - Access 2003: The provider is **"Microsoft.Jet.OLEDB.4.0"**. For 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.

ASCII - File [▶ 43]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"ASCII"**. →PLC: **eDBType_ASCII**.
-DBServer:	This option won't be needed.
-DBProvider:	This option won't be needed.
-DBUrl	DBUrl contains the path to the ASC-File oder TXT-File or CSV-File. e.g. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.asc")
-DBTable:	This option won't be needed. No table could be declared in ASCII-Files.

NET-MySQL Database [▶ 41]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"NET_MySQL"**. →PLC: **eDBType_NET_MySQL**.
-ODBC Driver:	This option won't be needed.
-Server name:	Contains the name of the Server or the IP-address of the host.
-Database name:	Contains the name of the database.
-Port:	Contains the port for the connection. (Standard 3306)
-Protocol:	This option won't be needed.
-Scheme:	This option won't be needed.
-Sequence:	This option won't be needed.
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

ODBC-MySQL Database [▶ 41]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"ODBC_MySQL"**. →PLC: **eDBType_ODBC_MySQL**.
-ODBC Driver:	Insert the name of the ODBC-driver. ("MySQL ODBC 3.51 Driver")
-Server name:	Contains the name of the Server or the IP-address of the host.
-Database name:	Contains the name of the database.
-Port:	Contains the port for the ODBC - connection
-Protocol:	This option won't be needed.
-Scheme:	This option won't be needed.
-Sequence:	This option won't be needed.
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

ODBC-PostgreSQL Database [▶ 44]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"ODBC_PostgreSQL"**. →PLC: **eDBType_ODBC_PostgreSQL**.
-ODBC Driver:	Insert the name of the ODBC-driver. ("PostgreSQL UNICODE")
-Server name:	Contains the name of the Server or the IP-address of the host.
-Database name:	Contains the name of the database.
-Port:	Contains the port for the ODBC - connection
-Protocol:	This option won't be needed.
-Scheme:	Contains the name of the used scheme.
-Sequence:	This option won't be needed.

ODBC-PostgreSQL Database [▶ 44]	
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

ODBC-DB2 Database [▶ 45]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"ODBC_DB2"**. →PLC: **eDBType_ODBC_DB2**.
-ODBC Driver:	Insert the name of the ODBC-driver. ("IBM DB2 ODBC DRIVER")
-Server name:	Contains the name of the Server or the IP-address of the host.
-Database name:	Contains the name of the database.
-Port:	Contains the port for the ODBC - connection
-Protocol:	Contains the name of the used protocol (TCPIP).
-Scheme:	Contains the name of the used scheme.
-Sequence:	This option won't be needed.
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

OCI-Oracle Database (Oracle Calling Interface) [▶ 42]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"OCI_Oracle"**. →PLC: **eDBType_OCI_Oracle**.
-ODBC Driver:	This option won't be needed.
-Server name:	Contains the name of the Server or the IP-address of the host.
-Database name:	Contains the name of the database / service name.
-Port:	Port for the connection (default:1521)
-Protocol:	Protocol for the connection (default: TCP)
-Scheme:	Contains the name of the used scheme.
-Sequence:	Contains the name of the used sequence.
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

ODBC-Oracle Database [▶ 42]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"ODBC_Oracle"**. →PLC: **eDBType_ODBC_Oracle**.
-ODBC Driver:	Insert the name of the ODBC-driver. ("Microsoft ODBC for Oracle")
-Server name:	This option won't be needed. (Only local connection)
-Database name:	Contains the name of the database.
-Port:	This option won't be needed.
-Protocol:	This option won't be needed.
-Scheme:	Contains the name of the used scheme.
-Sequence:	Contains the name of the used sequence.
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

ODBC-InterBase Database [▶ 46]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"ODBC_InterBase"**. →PLC: **eDBType_ODBC_InterBase**.
-ODBC Driver:	Insert the name of the ODBC-driver. ("Firebird/InterBase(r) driver")
-Server name:	Contains the name of the Server or the IP-address of the host.
-Database name:	Contains the name of the database.
-ClientDll:	Contains the path to the fbclient.dll.
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

ODBC-Firebird Database [▶ 47]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"ODBC_Firebird"**. →PLC: **eDBType_ODBC_Firebird**.
-ODBC Driver:	Insert the name of the ODBC-driver. ("Firebird/InterBase(r) driver")
-Server name:	Contains the name of the Server or the IP-address of the host.
-Database name:	Contains the name of the database.
-ClientDll:	Contains the path to the fbclient.dll.
-Table name:	Contains the name of the table in which you will read or write.
-UserId:	Contains the name of the user.
-Password:	Contains the password for the authentication.

XML Database [▶ 43]	
-DBValueType:	If you only log alphanumeric data types and Boolean choose **"Double"**. If you want to log structs and strings too, you must select **"Bytes"**.
-DBType:	You must choose **"XML"**→PLC: **eDBType_XML**.
-DBServer:	Contains the name of the database
-DBProvider:	This option won't be needed.
-DBUrl	DBUrl contains the path to the XML-file. e.g. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.xml") The XSD-file must be in the same directory and must have the same file name. e.g. ("C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.xsd")
-DBTable:	Contains the name of the table in which you will read or write.

6.5.2 Microsoft SQL Database

The values of the variables are saved in a Microsoft SQL database.
Compatible versions: Microsoft SQL Database 2000/2005/2008. Declarations see "Declarations different Databases" [▶ 35]

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
ID	bigint	no	IDENTITY(1,1)
Timestamp	datetime	no	
Name	ntext	no	
ValueType="Double"			
Value	float	no	
ValueType="Bytes"			
Value	varbinary	no	

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1. This functionality makes the IDENTITY property possible.
The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

The table is created with the following SQL command:

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

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

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

6.5.3 Microsoft Access Database

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

Database files of Access 2000 and Access 2003 (*.mdb) are as compatible as database files of Access 2007 (*.accdb).
You only must declare different provider in the XML - configuration file for these different filetypes. More info at "Declaration different databases" [▶ 35]

The variable values are saved in the following table structure.

Column name	Data type	Characteristic
ID	AutoNumber	Field Size:= "Long Integer"; New Values:= "Increment"
Timestamp	Date/Time	Format:= "General Date"
Name	Text	
ValueType="Double"		
Value	Number	Field Size:= "Double"

Column name	Data type	Characteristic
ValueType="Bytes"		
Value	OLE Object	

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1.
The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column
The **"Value"** column stores the value of the variable.

E_DBColumnTypes	MS Access	PLC Control
eDBColumn_BigInt	Integer4	DINT
eDBColumn_Integer	Integer2	INT
eDBColumn_SmallInt	Integer2	INT
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

Important!
Do not save the database on the Compact Flash Card in case of an embedded system.
Either use the database in RAM, i.e., do not save to the "Hard disk" folder, or save on a network folder. To many write cycles to the Compact Flash Card can shorten its service life.

6.5.4 SQL Compact Database

The values of the variables are saved in a Microsoft SQL Compact database.
Microsoft SQL Server 2005 Compact Edition is a compact database, ideal for embedding in mobile and desktop applications. It offers developers a programming model common with other editions of SQL Server for developing system dedicated and managed applications. This SQL Server requires relatively few resources, but nevertheless provides the necessary functionality for relational databases such as a robust data store, an optimized query processor and reliable, scalable connection functions.

Compatible version: Microsoft Compact SQL Database 3.5

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
ID	bigint	no	IDENTITY(1,1)
Timestamp	datetime	no	
Name	ntext	no	
ValueType = "Double"			
Value	float	no	
ValueType = "Bytes"			
Value	image	no	

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1. This functionality makes the IDENTITY property possible.
The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

The table is created with the following SQL command:

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

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

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

Important!
Do not save the database on the Compact Flash Card in case of an embedded system.
Either use the database in RAM, i.e. do not save to the "Hard disk" folder, or save on a network folder. To many write cycles to the Compact Flash Card can shorten its service life.

6.5.5 ODBC - MySQL Database

The values of the variables are saved in a MySQL database.

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
ID	INTEGER	no	IDENTITY(1,1)
Timestamp	DATETIME	no	
Name	VARCHAR(50)	no	
ValueType="Double"			
Value	DOUBLE	no	
ValueType="Bytes"			

Column name	Data type	Null permitted	Characteristic
Value	BLOB	no	

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1. This functionality makes the IDENTITY property possible.
The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

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

6.5.6 OCI / ODBC - Oracle Database

The values of the variables are saved in an Oracle Database.

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
ID	NUMBER	no	IDENTITY(1,1)
Timestamp	DATE	no	
Name	VARCHAR2	no	
ValueType="Double"			
Value	FLOAT	no	
ValueType="Bytes"			
Value	BLOB	no	

To get the functionality of an AutoID a sequence will be create at the scheme you use with the following attributes:

name: For example: "AUTO_INCREMENT_myTable$"

typ: "ascending"

minimum: "1"

maximum: "1.0E27"

intervall: "1"

cache: "no Cache"

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1. This functionality makes the sequence possible.
The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

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

6.5.7 ASCII - File

The values of the variables are saved in an ASCII file.
The values are written into the ASCII file separated by semicolons.
The file created can then be imported into other spreadsheet programs such as "Microsoft Excel", where it can be further processed.

The ASCII file has the following structure:

```
[Timestamp];[NAME];[VALUE]
[YYYY-MM-DD hh:mm:ss];[Variablename];[Variablevalue]
[YYYY-MM-DD hh:mm:ss];[Variablename];[Variablevalue]
[YYYY-MM-DD hh:mm:ss];[Variablename];[Variablevalue]
[YYYY-MM-DD hh:mm:ss];[Variablename];[Variablevalue]
```

The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

Important!
Do not save the database on the Compact Flash Card in case of an embedded system.
Either use the database in RAM, i.e. do not save to the "Hard disk" folder, or save on a network folder. To many write cycles to the Compact Flash Card can shorten its service life.

6.5.8 XML - Database

The variable values will be saved at a XML file. The structure of the database, tables and columns are defined at the XSD-file. With the function blocks FB_DBCreate [▶ 68] and FB_DBTableCreate [▶ 69] it is possible to create the XML file and the XSD file. Further information to work with XML files togehter with the TwinCAT3 Database Server you can find here: hier [▶ 48].

```
<?xml version="1.0" encoding="UTF-8"?>
<TestDB_XML xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:noNamespaceSchemaLocation="TestDB_XML.xsd">
  <myTable_Double>
```

```
    <row ID="1" Timestamp="2012-03-08T12:45:08" Name="TestValue1" Value="222.222" />
    <row ID="2" Timestamp="2012-03-08T12:45:14" Name="TestValue1" Value="222.222" />
    <row ID="3" Timestamp="2012-03-08T12:45:18" Name="TestValue1" Value="222.222" />
    <row ID="4" Timestamp="2012-03-08T12:45:22" Name="TestValue1" Value="222.222" />
    <row ID="5" Timestamp="2012-03-08T12:45:23" Name="TestValue1" Value="222.222" />
 </myTable_Double>
</TestDB_XML>
```

Table 3: The variable values are saved in the following table structure.

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

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1.
The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

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

Important!
Do not save the database on the Compact Flash Card in case of an embedded system.
Either use the database in RAM, i.e. do not save to the "Hard disk" folder, or save on a network folder. To many write cycles to the Compact Flash Card can shorten its service life.

6.5.9 ODBC - PostgreSQL Database

The values of the variables are saved in a PostgreSQL Database.

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
id	bigint	no	IDENTITY(1,1)
timestamp	timestamp	no	

Column name	Data type	Null permitted	Characteristic
name	text	no	
ValueType="Double"			
value	double precision	no	
ValueType="Bytes"			
value	BLOB	no	

To get the functionality of an AutoID a sequence will be create at the scheme you use with the following attributes:

name: "mytable_ID_seq"

raise step: "1"

minimum: "1"

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1. This functionality makes the sequence possible.
The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

E_DBColumnTypes	PostgreSQL	PLC Control
eDBColumn_BigInt	bigint	DINT
eDBColumn_Integer	integer	DINT
eDBColumn_SmallInt	smallint	INT
eDBColumn_TinyInt	smallint	INT
eDBColumn_Bit	bit	BYTE
eDBColumn_Money	money	LREAL
eDBColumn_Float	double precision	LREAL
eDBColumn_Real	real	REAL
eDBColumn_DateTime	timestamp	DT
eDBColumn_NText	text	STRING
eDBColumn_NChar	character	STRING
eDBColumn_Image	bytea	ARRAY OF BYTE
eDBColumn_NVarChar	character varying	STRING
eDBColumn_Binary	bytea	ARRAY OF BYTE
eDBColumn_VarBinary	bytea	ARRAY OF BYTE

6.5.10 ODBC - DB2 Database

The values of the variables are saved in a DB2 Database.

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
ID	BIGINT	no	IDENTITY (1,1)
Timestamp	TIMESTAMP	no	
Name	VARCHAR	no	
ValueType="Double"			
Value	DOUBLE	no	
ValueType="Bytes"			
Value	BLOB	no	

An AutoID is generated in the "ID" column. The value in this column is, in other words, always increased by 1. This functionality makes the IDENTITY property possible.
The "Timestamp" column stores the time at which the data record was saved.
The name of the variable is stored in the "Name" column.
The "Value" column stores the value of the variable.

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

6.5.11 ODBC - InterBase Database

The values of the variables are saved in an InterBase Database.

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
ID	BIGINT	no	IDENTITY(1,1)
Timestamp	TIMESTAMP	no	
Name	TEXT	no	
ValueType="Double"			
Value	FLOAT	no	
ValueType="Bytes"			
Value	BLOB	no	

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1.

The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

E_DBColumnTypes	InterBase	PLC Control
eDBColumn_BigInt	BIGINT	T_ULARGE_INTEGER (TcUtilities.lib)
eDBColumn_Integer	INTEGER	DINT
eDBColumn_SmallInt	SMALLINT	INT
eDBColumn_TinyInt	SMALLINT	INT
eDBColumn_Bit	CHAR(1)	BYTE
eDBColumn_Money	DEZIMAL(18,4)	LREAL
eDBColumn_Float	FLOAT	REAL

E_DBColumnTypes	InterBase	PLC Control
eDBColumn_Real	DOUBLE PRECISION	LREAL
eDBColumn_DateTime	Timestamp	DT
eDBColumn_NText	VARCHAR(254)	STRING
eDBColumn_NChar	CHAR(254)	STRING
eDBColumn_Image	BLOB	ARRAY OF BYTE
eDBColumn_NVarChar	VARCHAR(254)	STRING
eDBColumn_Binary	BLOB	ARRAY OF BYTE
eDBColumn_VarBinary	BLOB	ARRAY OF BYTE

6.5.12 ODBC - Firebird Database

The values of the variables are saved in an Firebird Database.

The variable values are saved in the following table structure.

Column name	Data type	Null permitted	Characteristic
ID	BIGINT	no	IDENTITY(1,1)
Timestamp	TIMESTAMP	no	
Name	TEXT	no	
ValueType="Double"			
Value	FLOAT	no	
ValueType="Bytes"			
Value	BLOB	no	

An AutoID is generated in the **"ID"** column. The value in this column is, in other words, always increased by 1.

The **"Timestamp"** column stores the time at which the data record was saved.
The name of the variable is stored in the **"Name"** column.
The **"Value"** column stores the value of the variable.

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

6.5.13 Additional information

6.5.13.1 Microsoft SQL Server hints

Logs of the Windows event log:

- Error Event „Report Server Windows service (SQLEXPRESS) can't connect to reporting server - database. “
  - At the SQL Configuration Manager at SQL Server 2005 services stop the SQL Server Reporting Services (SQLEXPRESS) and change the start mode to "Manual". The Reporting Service won't be needed from the TwinCAT Database Server.
- Information Event „'"Databasename"'-database will be started “
  - At the SQL Server Management Studio Express at Databases/"Databasename" rigth click -> properties and at options change the option "Automatic close" from "True" to "False". This option won't be needed, the TwinCAT Database Server will open and close the database automatically.

It is possible to blanket the logging into Windows event log. Then events from the SQL Server won't be logged. You can't choose which kind of event will be logged and which not.

- At the SQL Configuration Manager at SQL Server 2005 Services choose the SQL Server (SQLEXPRESS) and then right click -> properties. The "Advance" tab contains the subitem "Startup Parameters". The different parameters are separated with semicolons. Add the parameter -n and restart the service.

At this moment no events from the SQL Server will be logged to the Windows event log.

6.5.13.2 XML - Information

Use XML file as database with TF6420 Database Server [▶ 48]

Execute XPath querries at a XML file wiht the TF6420 Database Server [▶ 51]

You can find further information about XML-Schema here: http://www.w3.org/TR/xmlschema-0/

XML as Database

XSD-Schema for standard table structure

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

```
      <xsd:sequence minOccurs="1" maxOccurs="1">
        <xsd:element name="myTable_Double" type="myTable_Double_Type" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML file for standard table structure (Sample)

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

Datatypes for XML tables

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

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

 <xsd:simpleType name="ntext_80"> <!-- Length can be set individually. -->
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="80" />
   </xsd:restriction>
  </xsd:simpleType>

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

  <xsd:simpleType name="binary_1"> <!-- Length can be set individually. -->
   <xsd:restriction base="xsd:hexBinary">
     <xsd:maxLength value="1" />
   </xsd:restriction>
  </xsd:simpleType>

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

 <xsd:simpleType name="image_1"> <!-- Length can be set individually. -->
   <xsd:restriction base="xsd:hexBinary">
     <xsd:maxLength value="1" />
   </xsd:restriction>
  </xsd:simpleType>

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

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

  <xsd:simpleType name="nchar_50"> <!-- Length can be set individually.-->
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="50" />
   </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="nvarchar_50"> <!-- Length can be set individually.-->
   <xsd:restriction base="xsd:string">
     <xsd:maxLength value="50" />
   </xsd:restriction>
  </xsd:simpleType>

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

```
  </xsd:simpleType>

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

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

  <xsd:simpleType name="varbinary_1"> <!-- Length can be set individually.-->
   <xsd:restriction base="xsd:hexBinary">
      <xsd:maxLength value="1" />
    </xsd:restriction>
 </xsd:simpleType>
```

Datatype mapping XML => PLC

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

Creating/Reading of data records into/from an XML file

For creating data records, it is possible to use standard SQL commands. The SQL INSERT commands will be interpreted of the TwinCAT3 Database Server and creates the specified XML-Nodes for the used XML file. The SQL SELECT commands will be converted to XPath queries which will be executed at the used XML file.

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:

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

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

XML standard XPath function

XPath Types

3 different modes are supported to read values of an XML file...

- XPath<ATTR>
  - All attribute values of the selected XML tag will be returned to the PLC.
  - If an XML-Schema is available the attribute values will be converted to the defined data types.
  - If no XML-Schema is available all attribute values will be returned as T_MaxString
- XPath<TAG>
  - The inner text of the XML tag will be returned to the PLC.
  - If a XML-Schema is available the values will be converted to the defined data types.
  - If no XML-Schema is available all values will be returned as T_MaxString
- XPath<SUBTAG>
  - The inner text values of all XML subtags will be returned to the PLC.
  - If an XML-Schema is available, the values will be converted to the defined data types.
  - If no XML-Schema is available all values will be returned as T_MaxString

Samples

XML file:

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

XML schema:

```
<? xml version="1.0" encoding="utf-8" ?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
  <xs:element name="TestXML">
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="Node">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="SubNode1" type="xs:string" />
              <xs:element name="SubNode2" type="xs:short" />
              <xs:element name="SubNode3" type="xs:string" />
              <xs:element name="SubNode4" type="xs:double" />
              <xs:element name="SubNode5" type="xs:string" />
            </xs:sequence>
            <xs:attribute name ="attr1" type="xs:integer" use="required" />
            <xs:attribute name ="attr2" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Sample for XPATH<ATTR>

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

Returned structure if **no** XML schema is available.

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

Returned structure if a XML schema is available.

```
TYPE ST_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** XML schema is available: `SubNode2 : T_MaxString := '200';`

Returned value if an XML schema is available: `SubNode2 : INT := 200;`

Sample for XPATH<SUBTAG>

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

Returned structure if **no** XML schema is available.

```
TYPE ST_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 a XML schema is available.

```
TYPEYPE ST_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

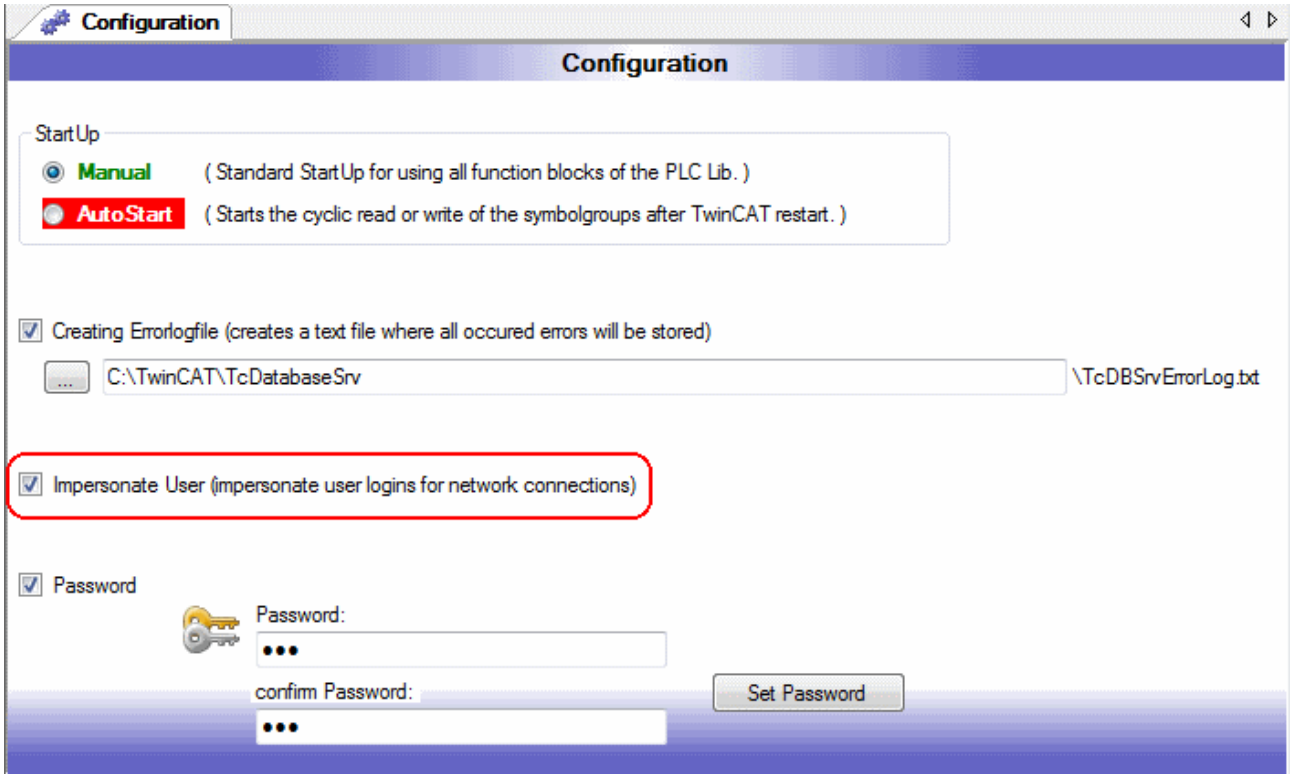
FB_DBRecordSelect_EX

FB_DBRecordArraySelect

6.6 Expert

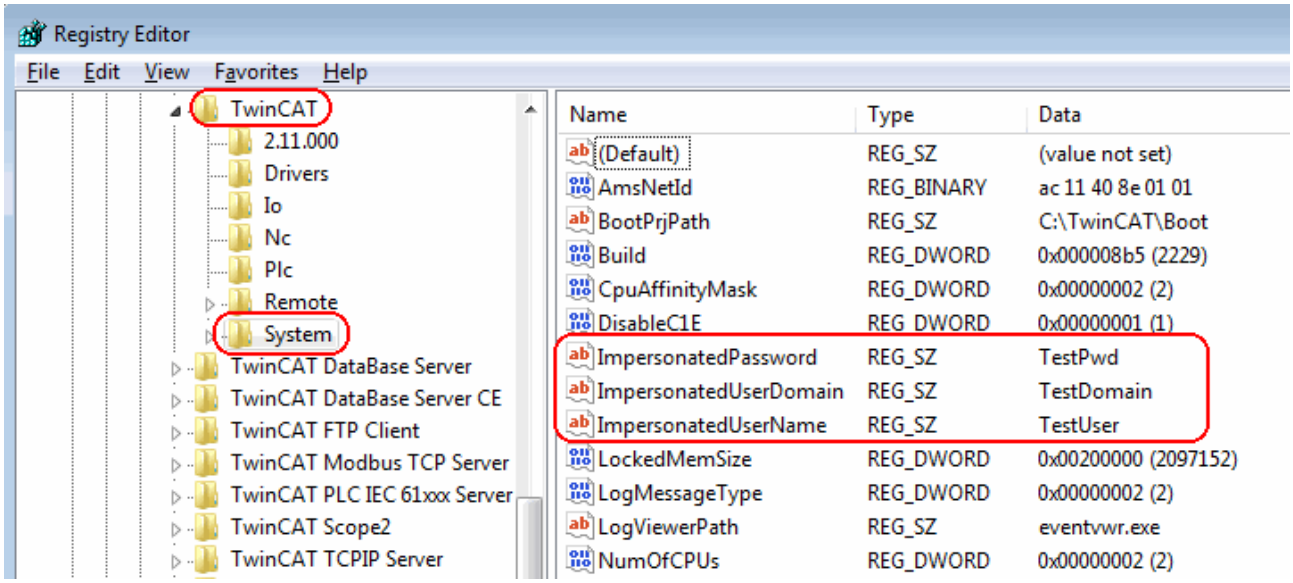
6.6.1 Impersonate option

With the option Impersonate it is possible to indicate values for user authentication to create a network connection to public directories.



The TwinCAT Database Server will be executed in a system process. This is the reason why the Database Server have no rights on a remote device. You can circumvent this problem if grant read an write rights to the user "guest" on the public folder. Please notice, that this method is very unsafe, because every user has rights at the public folders.

To solve this problem the TwinCAT Database server contains the "Impersonate"-option to logon at the remote device with specified user authentication values. You only have to write these values into the Registry like the following image illustrate.



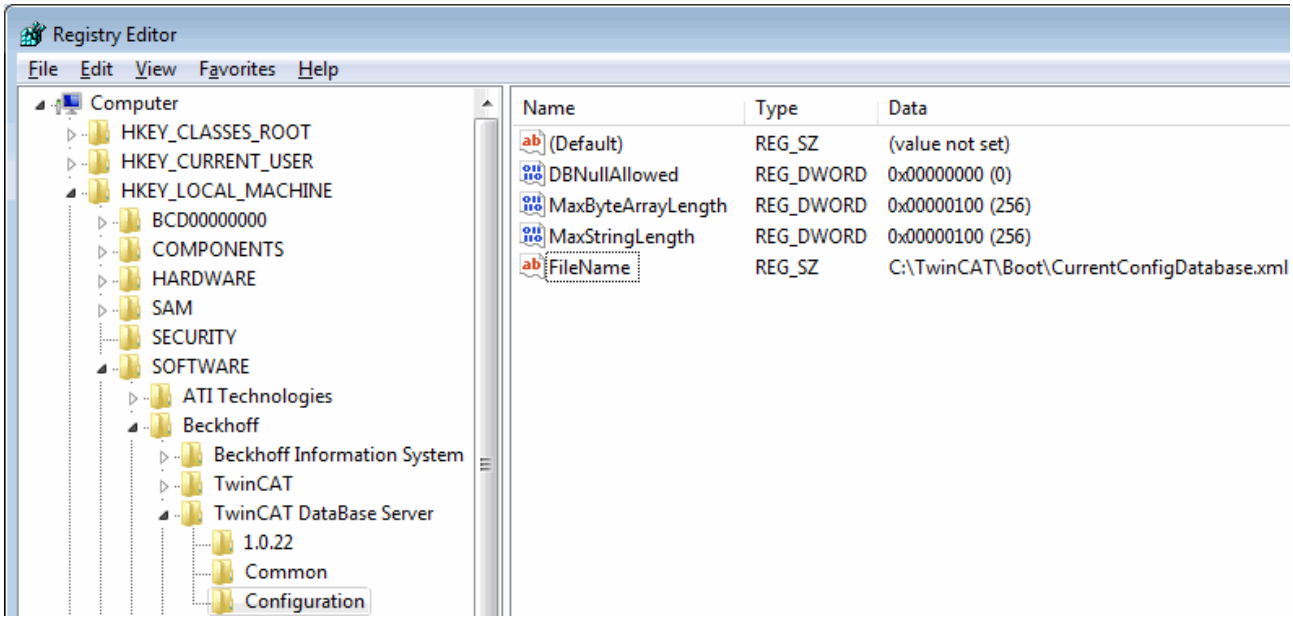
The following Keys must be created:
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\System] **"ImpersonatedPassword"**

Contains the password for the authentication.

->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\System] **"ImpersonatedUserDomain"**
Contains the name of the domain in which the user is logged on.
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT\System] **"ImpersonatedUserName"**
Contains the Name of the User.

6.6.2 Additional Registry configuration

There is an option to make basic settings for the TwinCAT Database Server. The key "DBNullAllowed" can be used to activate toleration of NULL values from the database. In this case the function blocks do not return an error if NULL values occur. In addition, maximum lengths for byte arrays and strings can be defined. These maximum lengths are used for the function blocks FB_DBRecordReturn, FB_DBRecordArrayReturn, FB_DBStoredProceduresRecordReturn and FB_DBStoredProceduresRecordArray.



The following keys can be adapted:
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT Database Server\Configuration]
"DBNullAllowed"

By setting a 1 at this key the allowing of DBNull values can be enabled. If NULL values occur and this key is not set, an error is returned to the respective function block.
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT Database Server\Configuration]
"MaxByteArrayLength"

Contains the maximum length of a byte array returned from the database to the PLC.
->[HKEY_LOCAL_MACHINE\SOFTWARE\Beckhoff\TwinCAT Database Server\Configuration]
"MaxStringLength"

Contains the maximum length of a string returned from the database to the PLC.

This function is supported from version 1.0.20.

6.6.3 XML configuration file

The TwinCAT DataBase Server is configured via an XML configuration file.
The settings of the configuration file are read once when the TwinCAT Database Server is started. Reading of the configuration can also be triggered from the PLC via a function block during TwinCAT DataBase Server runtime.

The configuration contains several sections:

- DataBases:
  Configuration of all "databases" including SQL database, ASCII file, ...
- AdsDevices:
  Configuration of all ADS devices (e.g. PLC runtime systems)
- SymbolGroups:
  Grouping of different "symbols" (e.g. PLC variables) associated with an ADS device into a logical group.
  A logical group can be configured for transporting data:
  - from a database to an ADS system,
  - from an ADS device to a database.

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

```
            <Symbol>
               <DBName>TESTVAR123</DBName>
               <Name>MAIN.TESTVAR123</Name>
               <Type>LREAL</Type>
               <IGroup>16448</IGroup>
               <IOffset>172536</IOffset>
               <BitSize>64</BitSize>
               <DBLogMode>3</DBLogMode>
            </Symbol>
         </Symbols>
      </SymbolGroup>
   </SymbolGroups>
</Configuration>
```

Notes regarding the XML configuration file:

- Tag „Log“:

Additional option: You can activate the error log mode for tests, to log occurred error descriptions into a text file "TcDBSrvErrorlog.txt".

- Tag „LogPath“:

Additional option: For tests it is possible to activate the error log mode see tag "Log". The path of the text file is declared in this tag.

- Tag „Impersonate“:

Additional option: To activate the "Impersonate" - option you have to add this tag.

- "StartUp" tag:

Start mode setting.
"Autostart" option : The server starts immediately with tasks specified in the configuration file (e.g. writing of ADS data into the database).
"Manual" option : The server initially remains passive and is triggered via the "FB_DBCyclicRdWrt" function block.

- "PwdInfos" tag:

Contains all passwords which are needed for the communication with the databases. All passwords are crypted and consequently nonreadable.

- "Databases" tag:

Configuration of the individual databases:

DBId each database is allocated a unique ID.
Database Type At present, **"Mobile Server"**, **"ASCII"**, **"Access"** and **"Sequal Server"** database types are supported.
ValueType Two storage methods are supported
- "Double" : Saves all the variable values as double values (structures and strings are not supported here);
- "Bytes" : Saves all the variable values as byte streams (structures and strings can also be logged in this way)

- "AdsDevices" tag:

Declaration of ADS devices.

Each runtime system is assigned a unique ID.

The ADS devices are identified through NetID, port and timeout.

"SymbolGoups" tag:

Definition of symbol groups: Specifies which database is linked with which ADS device. In addition the direction of the data flow is specified (from the TwinCAT system to the database or from the database to the TwinCAT system).

The cycle time determines at which intervals data are read from the database or written to the database.

- "Symbol" tag:

Definition of the individual symbols with the following parameters
DBName => Symbol name used in the database
Name => Symbol name used in the PLC
Optional: Type => Data type of the symbol
Optional: IGroup / IOffset => Storage location of the symbol
Optional: BitSize => Size of the symbol
DBLogMode => Optional: Specifies whether the symbols are checked cyclically or after a change.

The configuration file can be found under **"C:\TwinCAT\Boot\CurrentConfigDatabase.xml"** or, in embedded systems, under **"\Hard Disk\TwinCAT\Boot\CurrentConfigDatabase.xml"**

7 PLC API

Overview

The TcDatabase.lib library contains function blocks for controlling and configuring the TwinCAT database server.

Funktionsblöcke

Name	Description
FB_GetStateTcDatabase [▶ 59]	Call state information
FB_DBConnectionAdd [▶ 61]	Adds database connections to the XML configuration file
FB_DBAuthentificationAdd [▶ 81]	Adds authentication information for database connections to the XML configuration file
FB_DBOdbcConnectionAdd [▶ 62]	Adds ODBC database connections to the XML configuration file
FB_AdsDeviceConnectionAdd [▶ 64]	Adds Ads-device connections to the XML configuration file
FB_DBReloadConfig [▶ 60]	Reloads the XML configuration file
FB_GetDBXMLConfig [▶ 65]	Read all databaseconfiguration out of the XML-configuration file.
FB_GetAdsDevXMLConfig [▶ 66]	Read all Ads-deviceconfiguration out of the XML-configuration file.
FB_DBConnectionOpen [▶ 66]	Open a connection to a database
FB_DBConnectionClose [▶ 67]	Close a connection to a database
FB_DBCreate [▶ 68]	Creates a new database
FB_DBTableCreate [▶ 69]	Creates a table with any desired table structure
FB_DBRead [▶ 71]	Reads one value out of the database
FB_DBWrite [▶ 72]	Writes one variable value, with timestamp, into a database
FB_DBCyclicRdWrt [▶ 70]	Starts or stops the logging/writing of variables
FB_DBRecordSelect [▶ 83]	Reads a data record out of a table
FB_DBRecordSelect_EX [▶ 85]	Reads a data record out of a table (commandlength <= 10000 symbols)
FB_DBRecordArraySelect [▶ 76]	Reads some data records out of a table
FB_DBRecordInsert [▶ 82]	Creates a new data record
FB_DBRecordInsert_EX [▶ 75]	Creates a new data record (commandlength <= 10000 symbols)
FB_DBRecordDelete [▶ 74]	Deletes a data record from a table
FB_DBStoredProcedure [▶ 78]	Execute Stored Procedures.
FB_DBStoredProcedureRecordReturn [▶ 86]	Execute Stored Procedures and return a data record.
FB_DBStoredProcedureRecordArray [▶ 79]	Execute Stored Procedures and return a count of data records.

Funktionen

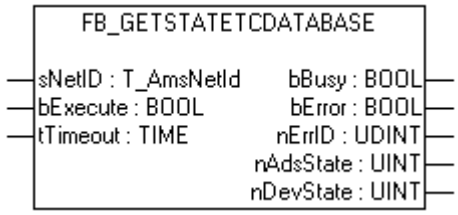
Name	Description
F_GetVersionTcDatabase [▶ 87]	Call version information.

Datentypen

Name
ST_DBColumnCfg [▶ 88]
ST_DBXMLCfg [▶ 88]
ST_ADSDevXMLCfg [▶ 88]
ST_DBSQLError [▶ 89]
ST_DBParameter [▶ 89]
E_DbColumnTypes [▶ 90]
E_DBTypes [▶ 91]
E_DBValueType [▶ 91]
E_DBWriteModes [▶ 91]
E_DBParameterTypes [▶ 91]

7.1 Function blocks

7.1.1 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

bExecute : The command is executed with the rising edge.

tTimeout : States the time before the function is cancelled.

VAR_OUTPUT

```
VAR_OUTPUT
    bBusy       : BOOL;
    bError      : BOOL;
    nErrID      : UDINT;
    nAdsState   : 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 : Supplies the ADS Error Code [▶ 132] when 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 is started
- 2 = The cyclic read/write is started

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.2 FB_DBReloadConfig

With the FB_DBReloadConfig function block the XML configuration file can be reloaded.
If you have change the XML configuration file, you have to manifest these changes to the Database Server with the FB_DBReloadConfig.

VAR_INPUT

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

sNetID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

bExecute : The command is executed with the 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.

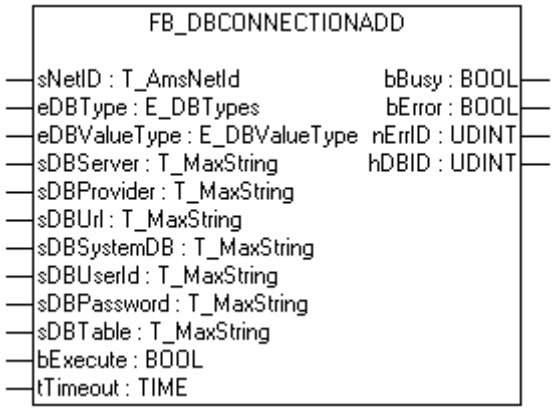
bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.3 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 : Is a string containing the AMS network identifier of the target device to 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 at "Access Databases". Contains the path to the MDW-file.

sUserId : Indicates the username for the registration.

sPassword : Contains the password

sDBTable : Gives the name of the table into which the values are to be written.

bExecute : The command is executed with the rising edge.

tTimeout : States 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.

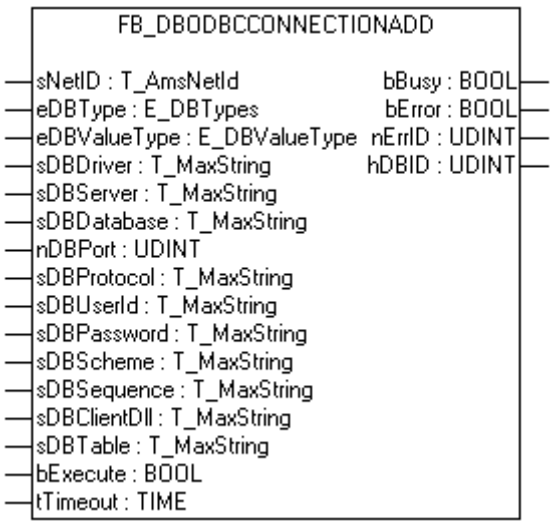
nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

hDBID : Returns the ID of the database.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.4 FB_DBOdbcConnectionAdd



The function block FB_DBOdbcConnectionAdd permits additional ODBC - database connections to be added 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 : Is a string containing the AMS network identifier of the target device to 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 : Gives the name of the ODBC - driver.

sDBServer : Provides the name of the server.

sDBDatabase : Gives the name of the database.

nDBPort : Gives the port of the ODBC-connection.

sDBProtocol : Contains the name of the used protocol (TCPIP).

sDBUserId : Indicates the username for the registration.

sDBPassword : Contains the password.

sDBScheme : Contains the name of the scheme.

sDBSequence : Contains the name of the sequence for the "autoID".(Only for Oracle DBs)

sDBClientDll : Contains the path to the fbclient.dll.(Only for Firebird/Interbase DBs)

sDBTable : Gives the name of the table into which the values are to be written.

bExecute : The command is executed with the rising edge.

tTimeout : States 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.

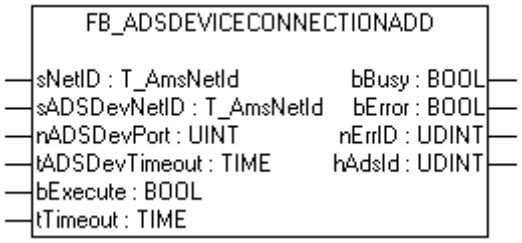
nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

hDBID : Returns the ID of the database.

Requirements

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

7.1.5 FB_AdsDeviceConnectionAdd



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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

sADSDevNetID : Is a string containing the AMS network identifier of the added Ads-device.

nADSDevPort : Indicates the Ams-port of the added Ads-device.

tADSDevTimeout : Contains the timeout of the added Ads-device.

bExecute : The command is executed with the rising edge.

tTimeout : States the time before the function is cancelled.

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 : Supplies the ADS Error Code [▶ 132] when the bError output is set.

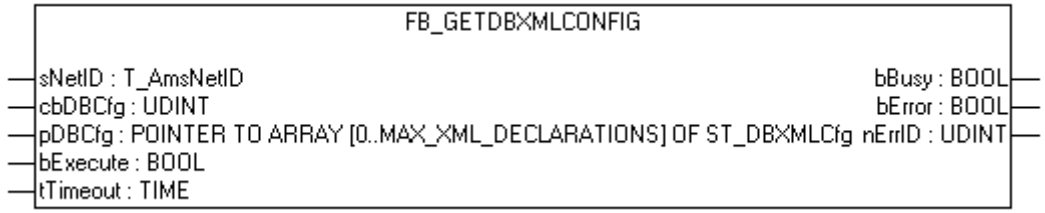
hAdsId : Returns the ID of the ADS-Device.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	CX (ARM)	

7.1.6 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

cbDBCfg : Returns the length of the array in which the configurations will be write.

pDBCfg : Provides the pointer address of the array in which the configurations will be write.

bExecute : The command is executed with the rising edge.

tTimeout : States 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.

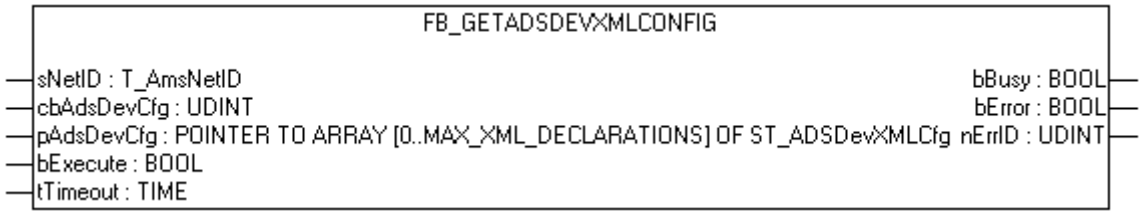
bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.7 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

cbAdsDevCfg : Returns the length of the array in which the configurations will be write.

pAdsDevCfg : Provides the pointer address of the array in which the configurations will be write.

bExecute : The command is executed with the rising edge.

tTimeout : States 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.

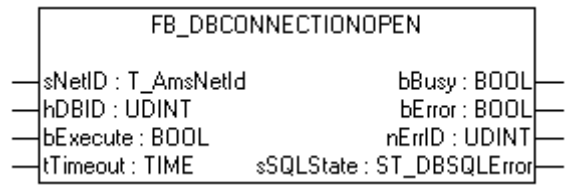
bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.8 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Indicates the ID of the database to be used.

bExecute : The command is executed with the rising edge.

tTimeout : States 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.

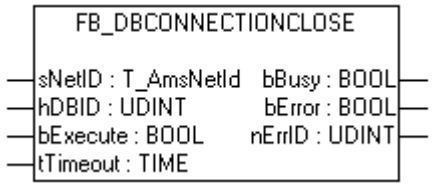
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.9 FB_DBConnectionClose



You can close connections to databases with this function block FB_DBConnectionOpen. If you have opened a connection to a database before, it is important to close this connection after using.

VAR_INPUT

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

sNetID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Indicates the ID of the database to be used.

bExecute : The command is executed with the rising edge.

tTimeout : States 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.

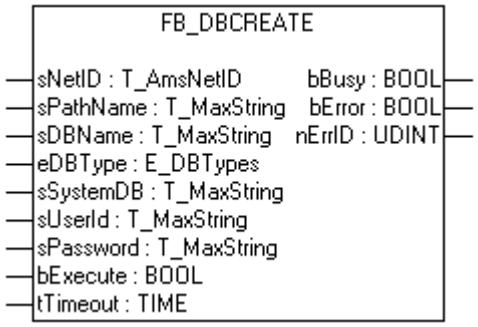
bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.10 FB_DBCreate



The FB_DBCreate function block allows databases to be created. MS SQL databases, MS SQL Compact databases, MS Access databases and XML databases can be created with this FB.

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 : Is a string containing the AMS network identifier of the target device to 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 at "Access Databases". Contains the path to the MDW-file.

sUserId : Indicates the username for the registration.

sPassword : Contains the password.

bExecute : The command is executed with the 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

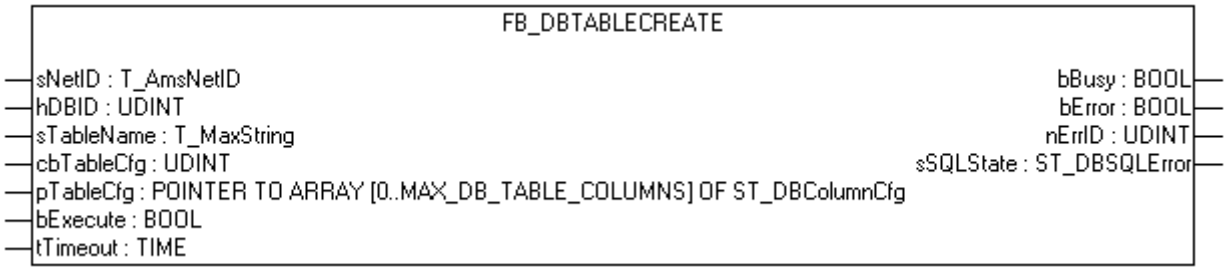
nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

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 v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.11 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Is the 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 the 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.

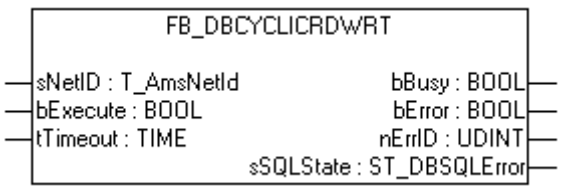
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv_Error_Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.12 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

bExecute : A rising edge starts the read/write cycle, while a falling edge stops it.

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.

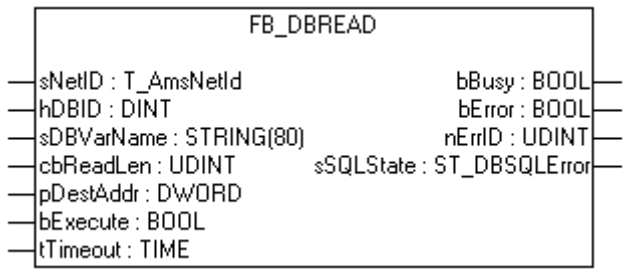
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.13 FB_DBRead



The FB_DBRead allows values to be read from a database.

The function block searches the database table in the "Name" column for the specified sDBVarName and then outputs the corresponding value from the "Value" column. If the sDBVarName searched for is present several times in the database table, the first data record found is output.

VAR_INPUT

```
VAR_INPUT
    sNetID          : T_AmsNetId;
    hDBID           : DINT;
    sDBVarName      : STRING(80);
    cbReadLen       : UDINT;
    pDestAddr       : DWORD;
```

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

sNetID : Is a string containing the AMS network identifier of the target device to 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 the rising edge.

tTimeout : States 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.

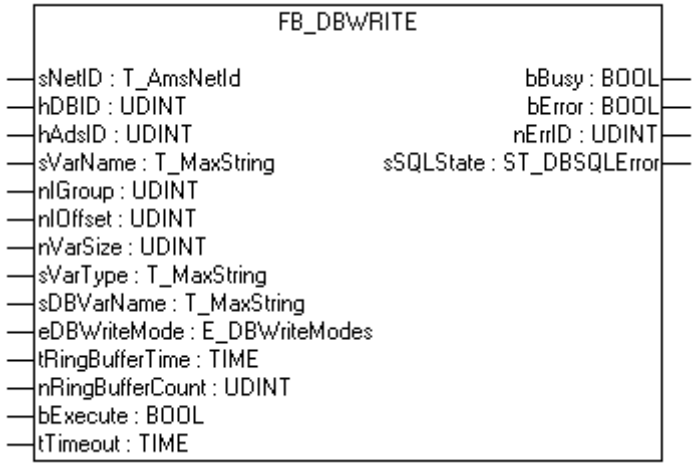
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.14 FB_DBWrite



The FB_DBWrite function block can be used to write the values of individual variables into databases. The table structure have to contain the columns "Timestamp", "Name" and "Value". see SQL Compact Database [▶ 40]. To use the function block, you have to declare the required database and ADS device 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Is the ID of the database to be used.

hAdsID : This is the 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 : Determines if the new records will be added or if the existing records will be updated.

tRingBufferTime : Contains the maximum age of datasets in a table. (only at Ringbuffer_WriteMode)

nRingBufferCount : Contains the maximum count of datasets in a table. (only at Ringbuffer_WriteMode)

bExecute : The command is executed with the rising edge.

tTimeout; : States 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 : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

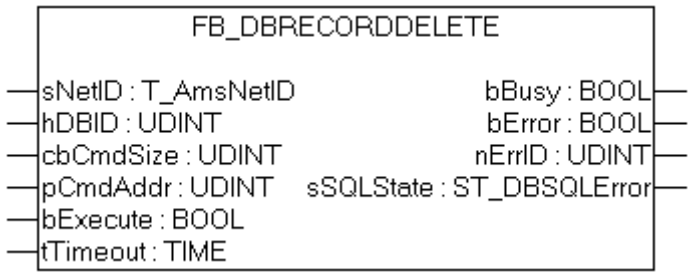
Logging of values from an ADS-device (not for BC9000)	Logging of values from a BC9000

`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 v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.15 FB_DBRecordDelete



The FB_DBRecordDelete function block allows to delete data records out of a database. The length of the SQL - command could be till 10000 symbols. For using this function block you have to declare the database, you want to delete from, in the XML - configuration file.

VAR_INPUT

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

sNetID : Is a string containing the AMS network identifier of the target device to 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 executed INSERT command.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

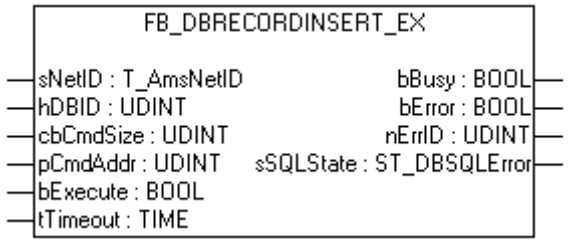
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.16 FB_DBRecordInsert_EX



The FB_DBRecordInsert_EX function block allows individual data records to be written into a database. The length of the SQL - command could be till 10000 symbols. For using this function block you have to declare the database, you want to write to, in the XML - configuration file.

VAR_INPUT

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

sNetID : Is a string containing the AMS network identifier of the target device to 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 executed INSERT command.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

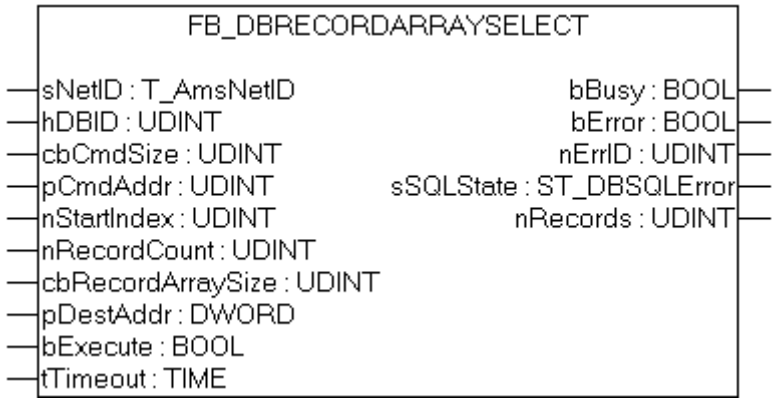
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.17 FB_DBRecordArraySelect



The FB_DBRecordArraySelect allows some individual data records to be read from a database. The length of the SQL-command could be till 10000 Symbols.

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

sNetID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Indicates the ID of the database to be used.

cbCmdSize : Indicates the size of the SELECT command.

pCmdAddr : Pointer to the executed SELECT command.

nStartIndex : Gives the index of the first data record that is to be read.

nRecordCount : Gives the count of data records that are to be read.

cbRecordArraySize : Provides the size of an arraystructure in bytes.

pDestAddr : Provides the address of the arraystructure into which the data records are to be written.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

nRecords : Returns the number of data records.

Example in ST:

Because the table out of which a data record is to be read has the following structure...

Column name	Data type
ID	bigint
Timestamp	datetime
Name	nvarchar(80
Value	float

... a PLC structure must be created having a comparable structure.

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

To get the data type T_ULARGE_INTEGER, you have to add the library TcUtilities.lib to the PLC-Program

For ARM - processors the order of the data types is different and you have to add a "Dummy-BYTE" to the struct because of the different byte alignment at ARM - processors.

```
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
```

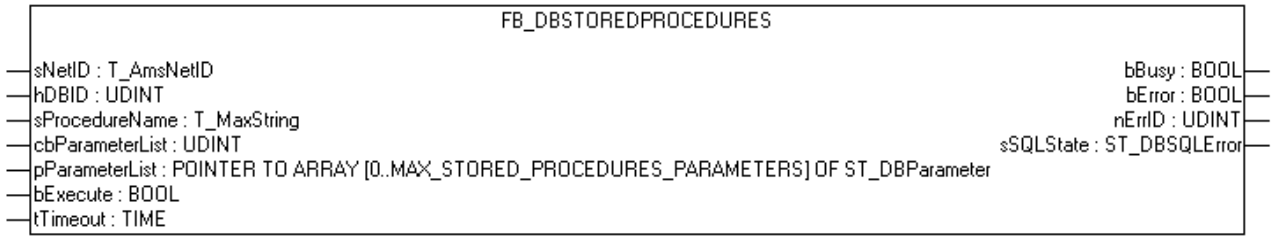
PLC program

```
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 v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.18 FB_DBStoredProcedures



With this function block FB_DBStoredProcedures you are able to start stored procedures. It is possible to declare parameters which will be 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Is the ID of the database to be used.

sProcedureName : Provides the name of the stored procedure.

cbParameterList : Provides the size of a parameter list in bytes.

pParameterList : Provides the address of a parameter list.

bExecute : The command is executed with the 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

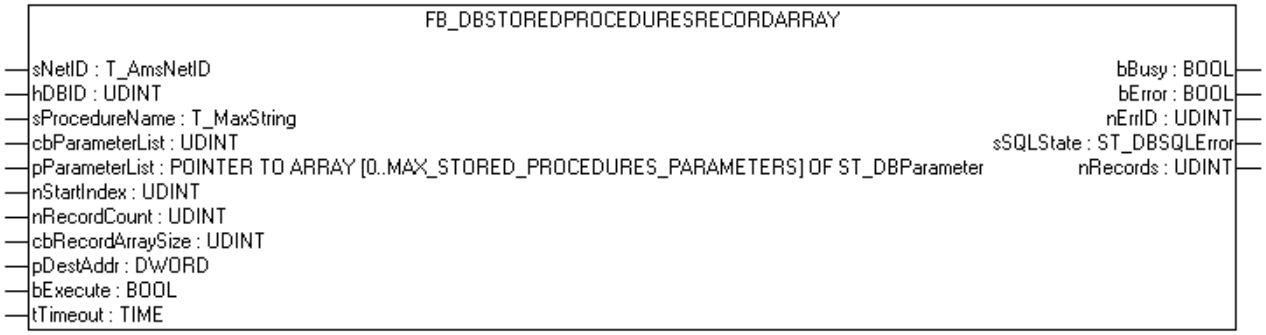
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv_Error_Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib (from TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

7.1.19 FB_DBStoredProceduresRecordArray



With this function block FB_DBStoredProceduresRecordArray you can start stored procedures which return data records. The difference between the FB_DBStoredProcedureRecordReturn and this function block is, the FB_DBStoredProcedureRecordArray can return more than one record with one execution. It is possible to declare parameters which will be 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        : DWORD;
    bExecute        : BOOL;
    tTimeout        : TIME          := T#15s;
END_VAR
```

sNetID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Is the ID of the database to be used.

sProcedureName : Provides the name of the stored procedure.

cbParameterList : Provides the size of a parameter list in bytes.

pParameterList : Provides the address of a parameter list.

nStartIndex : Gives the index of the first data record that is to be read.

nRecordCount : Gives the count of data records that are to be read.

cbRecordArraySize : Provides the size of an array structure in bytes.

pDestAddr : Provides the address of the array structure into which the data records are to be written.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv_Error_Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

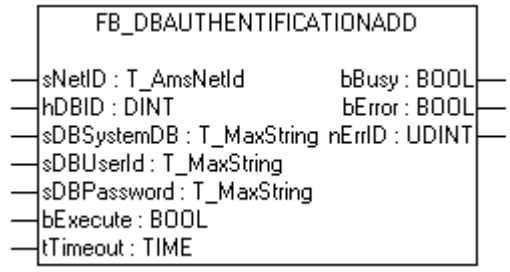
nRecords : Returns the number of data records.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib (from Version 1.0.17)
TwinCAT v2.10.0	CX (ARM)	

7.1.20 Obsolete

7.1.20.1 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 : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Indicates the ID of the database to be used.

sSystemDB : Only at "Access Databases". Contains the path to the MDW-file.

sUserId : Indicates the username for the registration.

sPassword : Contains the password.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

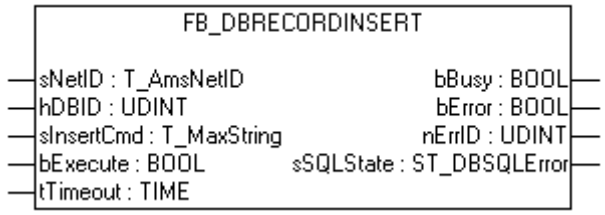
bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] when the bError output is set.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.20.2 FB_DBRecordInsert



The FB_DBRecordInsert function block allows individual data records to be written into a database. For using this function block, you have to declare the database, you want to write to, in the XML - configuration file.

VAR_INPUT

```
VAR_INPUT
    sNetID      : T_AmsNetId;
    hDBID       : UDINT;
    sInsertCmd  : T_MaxString;
    bExecute    : BOOL;
    tTimeout    : TIME;
END_VAR
```

sNetID : Is a string containing the AMS network identifier of the target device to 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 the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

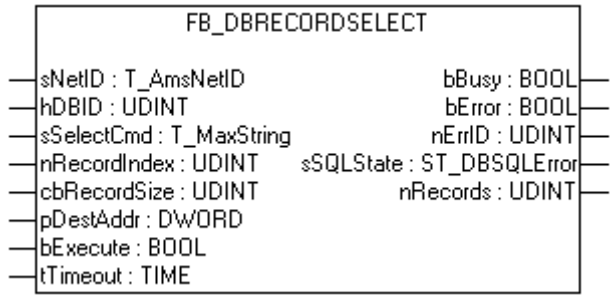
nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv_Error_Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.20.3 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 : Is a string containing the AMS network identifier of the target device to 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 : Provides the address of the structure into which the data record is to be written.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv Error Codes [▶ 135] when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

nRecords : Returns the number of data records.

Example in ST:

Because the table out of which a data record is to be read has the following structure...

Column name	Data type
ID	bigint
Timestamp	datetime
Name	ntext
Value	float

... a PLC structure must be created having a comparable structure.

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

To get the data type T_ULARGE_INTEGER, you have to add the library TcUtilities.lib to the PLC-Program

For ARM - processors the order of the data types is different and you have to add a "Dummy-BYTE" to the struct because of the different byte alignment at ARM - processors.

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

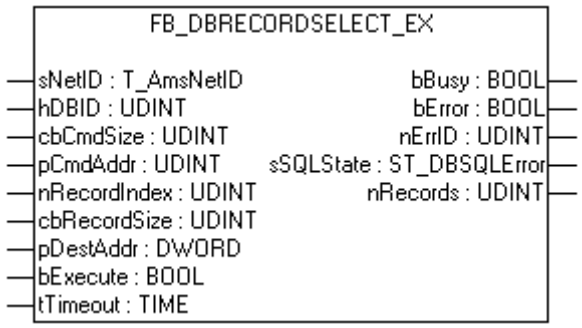
PLC program

```
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 v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.20.4 FB_DBRecordSelect_EX



The FB_DBRecordSelect_EX allows individual data records to be read from a database. The length of the SQL-command could be till 10000 Symbols.

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

sNetID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Indicates the ID of the database to be used.

cbCmdSize : Indicates the size of the SELECT command.

pCmdAddr : Pointer to the executed SELECT command.

nRecordIndex : Gives the index of the data record that is to be read.

cbRecordSize : Provides the size of a data record in bytes.

pDestAddr : Provides the address of the structure into which the data record is to be written.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 if "bBusy" remains TRUE.

bError : Becomes TRUE, as soon as an error occurs.

nErrID : Supplies the ADS Error Code [▶ 132] or the TcDatabaseSrv_Error_Codes [▶ 135] when the bError output is set.

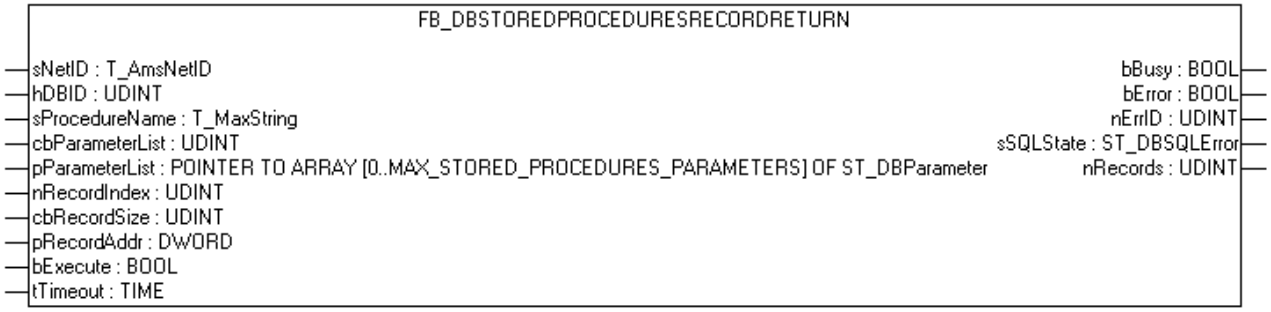
sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

nRecords : Returns the number of data records.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.1.20.5 FB_DBStoredProceduresRecordReturn



With this function block FB_DBStoredProcedures you are able to start stored procedures which return a data record. It is possible to declare parameters which will be 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     : DWORD;
    bExecute        : BOOL;
    tTimeout        : TIME          := T#15s;
END_VAR
```

sNetID : Is a string containing the AMS network identifier of the target device to which the ADS command is directed.

hDBID : Is the ID of the database to be used.

sProcedureName : Provides the name of the stored procedure.

cbParameterList : Provides the size of a parameter list in bytes.

pParameterList : Provides the address of a 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.

pDestAddr : Provides the address of the structure into which the data record is to be written.

bExecute : The command is executed with the rising edge.

tTimeout : States 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 : Supplies the ADS Error Code or the TcDatabaseSrv_Error_Codes when the bError output is set.

sSQLState : Supplies the SQL error code [▶ 89] of the specified database type.

nRecords : Returns the number of data records.

Requirements

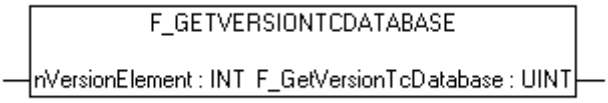
Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib (from TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

Also see about this

- ADS Return Codes [▶ 132]
- Internal Errorcodes of the TwinCAT Database Server [▶ 135]

7.2 Functions

7.2.1 F_GetVersionTcDatabase



This function can be used to read PLC library version information.

FUNCTION F_GetVersionTcDatabase: UINT

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

nVersionElement : Version element to be read. Possible parameters:

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

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3 Data types

7.3.1 ST_DBColumnCfg

```
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 system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.2 ST_DBXMLCfg

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

sDBName : Contains the name of the database.

sDBTable : Contains the name of the table.

nDBID : Returns the ID of the database.

eDBType : Contains the type of the database.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.3 ST_ADSDevXMLCfg

```
TYPE ST_ADSDevXMLCfg :
STRUCT
    sAdsDevNetID    : T_AmsNetID;
    tAdsDevTimeout  : TIME;
    nAdsDevID       : DINT;
```

```
    nAdsDevPort     : UINT;
END_STRUCT
END_TYPE
```

sAdsDevNetID : Is a string containing the AMS network identifier of the Ads-device.

tAdsDevTimeout : Contains the timeout of the Ads-device.

nAdsDevID : Returns the ID of the Ads-device.

nAdsDevPort : Indicates the Ams-port of the Ads-device.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.4 ST_DBSQLError

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

sSQLState : Contains the 5 character error code which follows the SQL ANSI standard.

nSQLErrorCode : Contains the database specific error code .

If no error occured the structure contains the following values.:
sSQLState := **'00000'**;
nSQLErrorCode := **0**;

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.5 ST_DBParameter

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

sParameterName : Contains the name of the parameter.

cbParameterValue : Contains the size of the used variable in bytes.

pParameterValue : Contains the pointer address of the used variable.

eParameterDataType : Gives the data type [▶ 90] of the parameter.

pParameterValue : Gives the type [▶ 91] of the parameter.

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 system type	PLC libraries to be linked
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib (from TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

7.3.6 E_DbColumnTypes

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

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.7 E_DBTypes

```
TYPE E_DBTypes :
(
    eDBType_Mobile_Server   := 0,
    eDBType_Access          := 1,
    eDBType_Sequal_Server   := 2,
    eDBType_ASCII           := 3,
    eDBType_ODBC_MySQL      := 4,
    eDBType_ODBC_PostgreSQL := 5,
    eDBType_ODBC_Oracle     := 6,
    eDBType_ODBC_DB2        := 7,
    eDBType_ODBC_InterBase  := 8,
    eDBType_ODBC_Firebird   := 9,
    eDBType_XML             := 10,  (*not supported*)
    eDBType_OCI_Oracle      := 11
);
END_TYPE
```

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.8 E_DBValueType

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

Requirements

Development Environment	Target System	PLC libraries to include
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.9 E_DBWriteModes

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

);
END_TYPE
```

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

7.3.10 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 system type	PLC libraries to be linked
TwinCAT v2.10.0	PC oder CX (x86)	TcDatabase.Lib (from TcDatabaseSrv Version 1.0.13)
TwinCAT v2.10.0	CX (ARM)	

7.4 Constants

7.4.1 Global Variables

```
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 System	PLC libraries to include
TwinCAT v2.10.0	PC or CX (x86)	TcDatabase.Lib
TwinCAT v2.10.0	CX (ARM)	

8 Samples

8.1 Quick Start

The following PDF-Document contains the handout from a workshop with the topic TwinCAT Database Server.

This handout explain in detail how to work with the XML-configuration file editor and shows the function of several function blocks out of the PLC.

Download as PDF: Sample 9 "Workshop Handout" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407900555/.pdf

Technical Workshop

Topic: TwinCAT Database Server

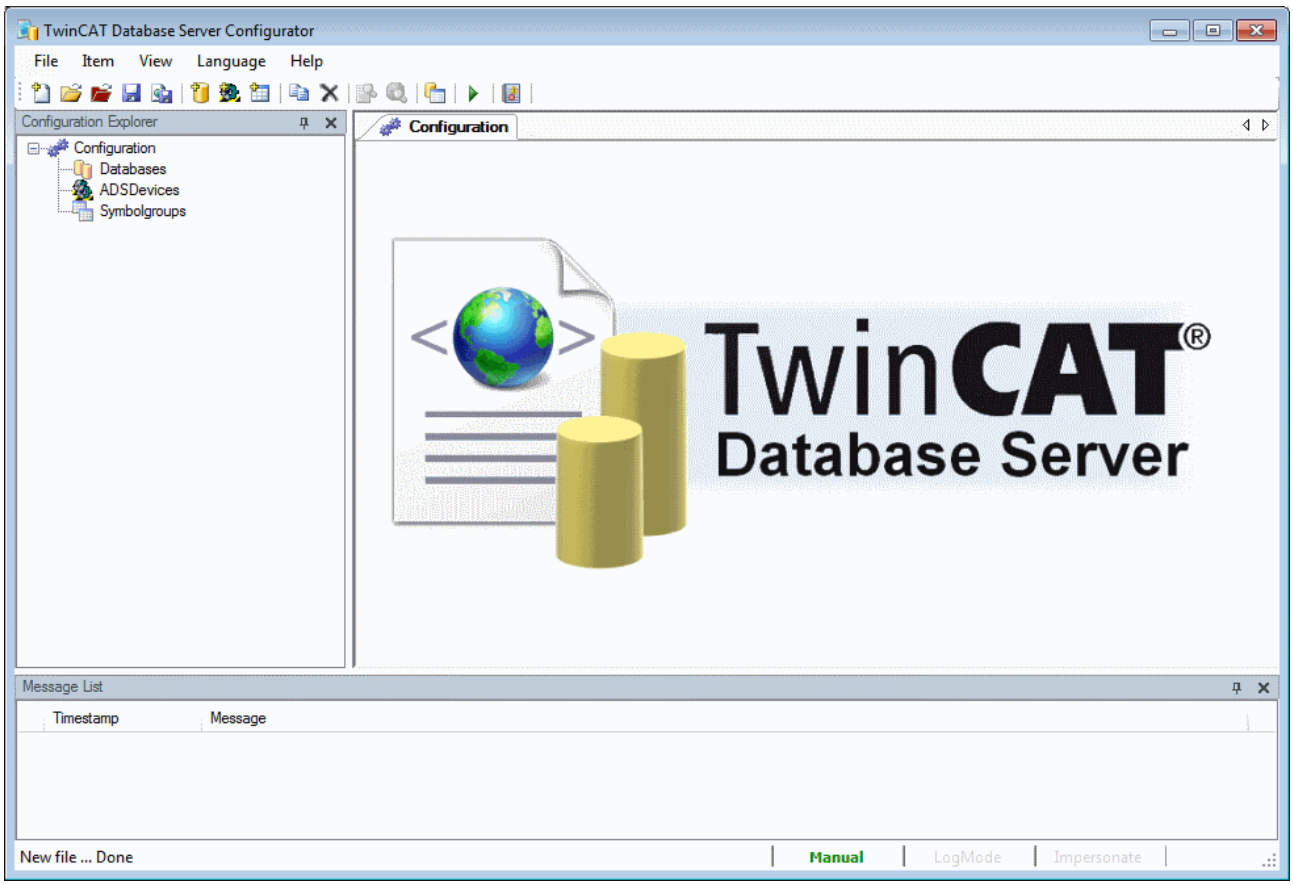
Step by step introduction for hands on with TwinCAT Database Server.

1. Foreword

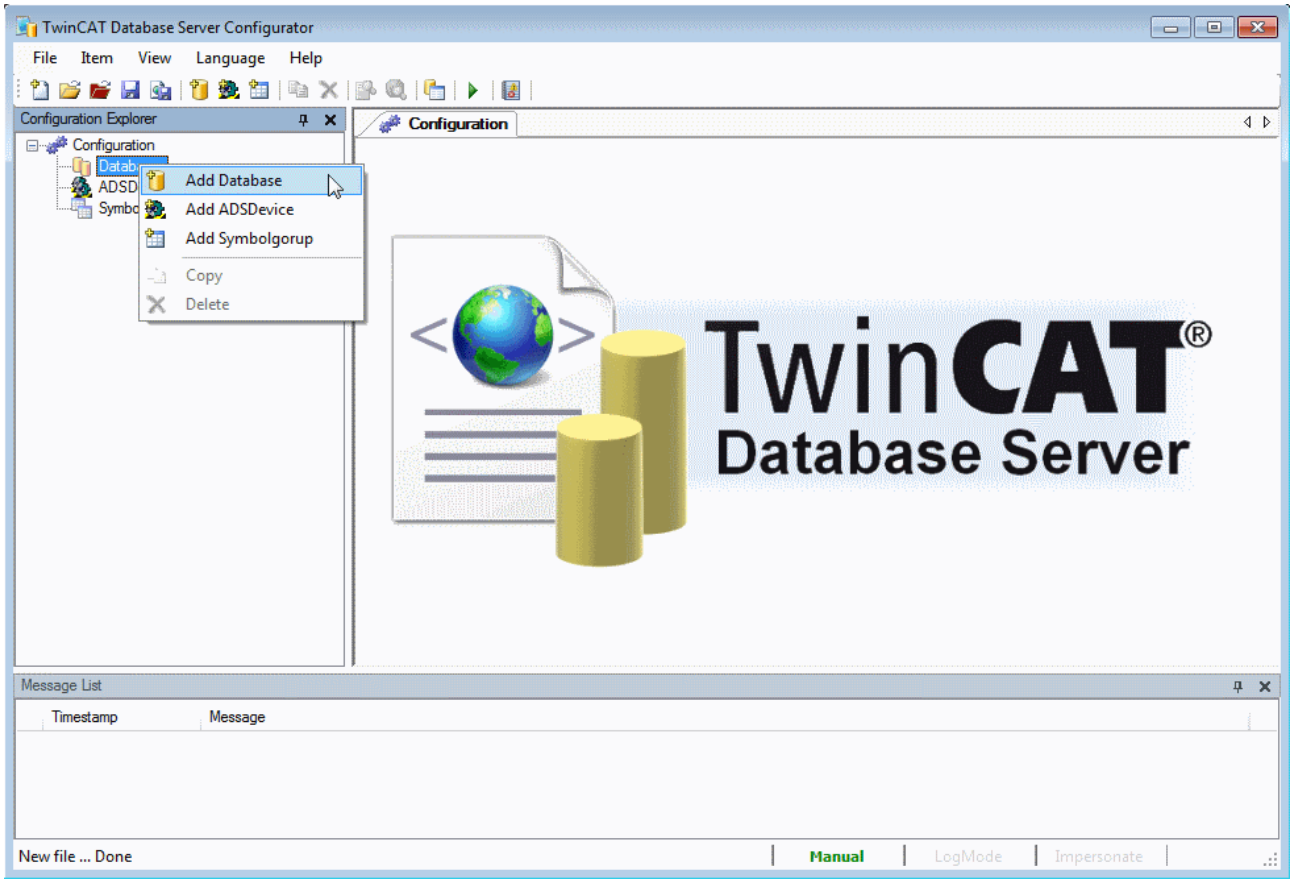
There are two possibilities to configure the TwinCAT Database Server - on the one hand out of the PLC Control and on the other hand with TwinCAT Database Server XML Configuration File Editor. Today we discuss a small example for the configuration with XML Editor, but there is also an instruction for the second way for testing yourself.

2. DataBaseServer Configuration with XML Editor

Open the Editor under Start -> All Programs -> TwinCAT System -> TwinCAT DataBase Server -> XML Configuration File Editor



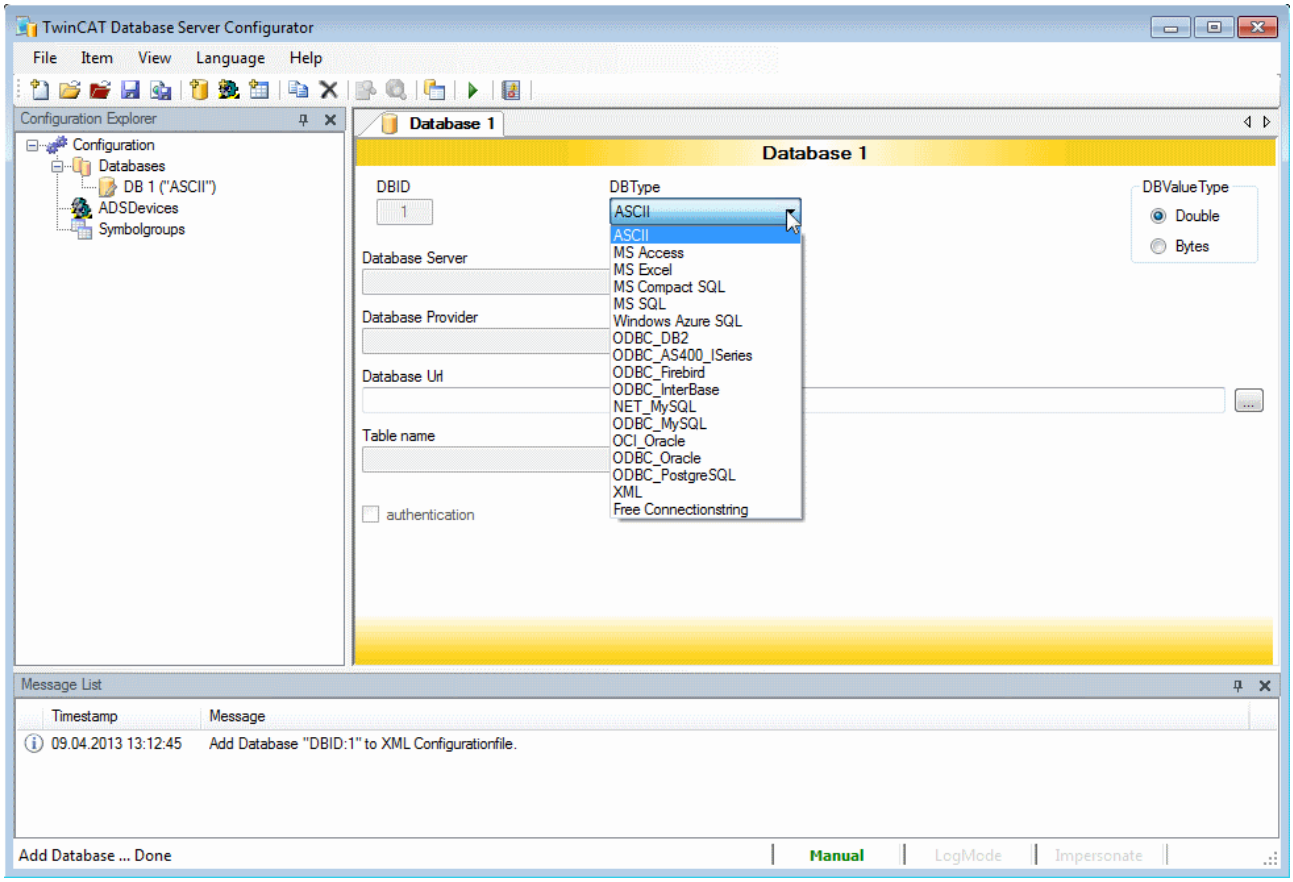
At first you have to add a new database. Right click on Database as it is shown in the picture or by the symbols in the tool bar.



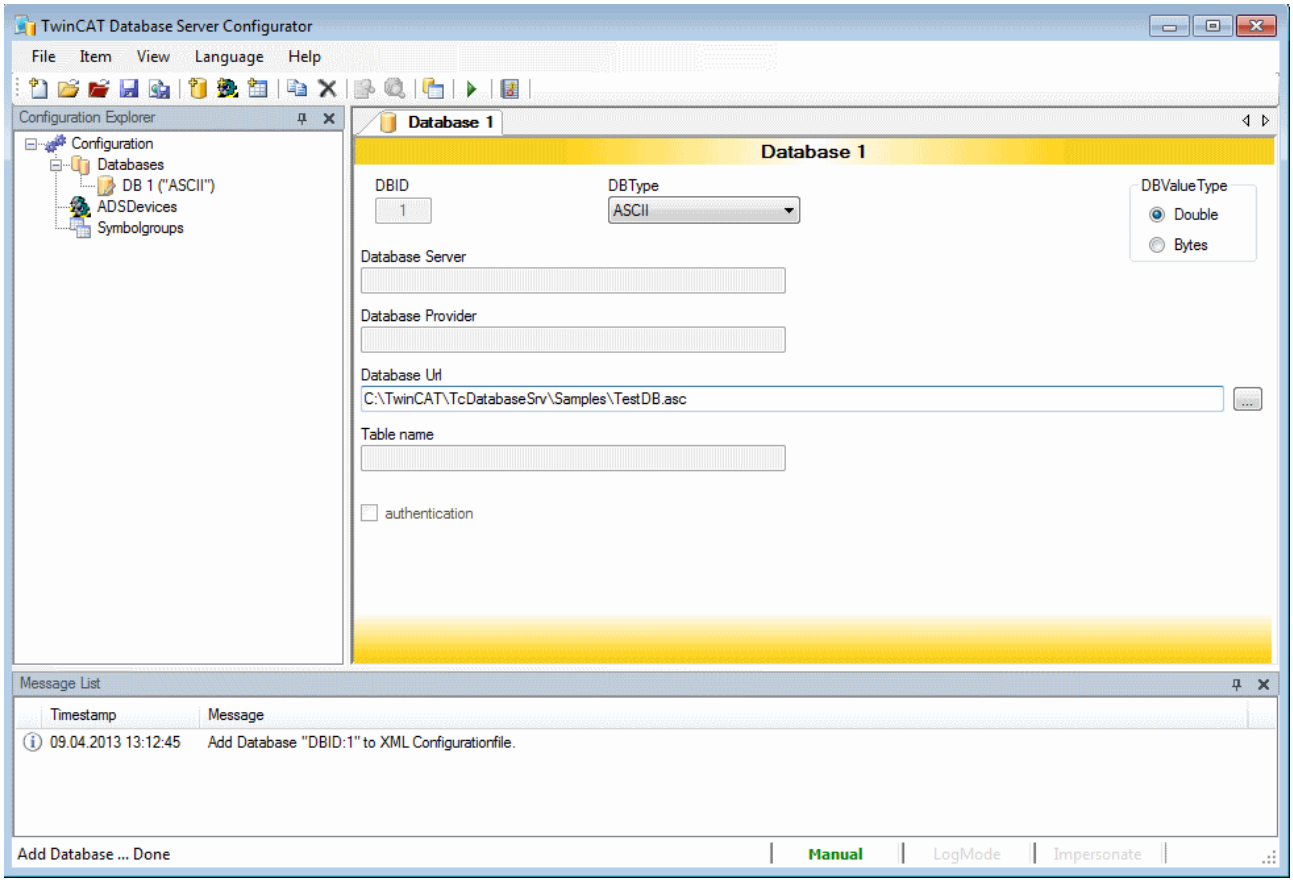
In the next step you must configure your database. Not everybody has got the great databases on his system, that's why we choose the ASCII database.
You can find the declaration of all different database types, which we support, in the Information System.

mk:@MSITStore:tcdbserver.chm/html/tcdbserver_dbdeclaration.htm [▶ 93]

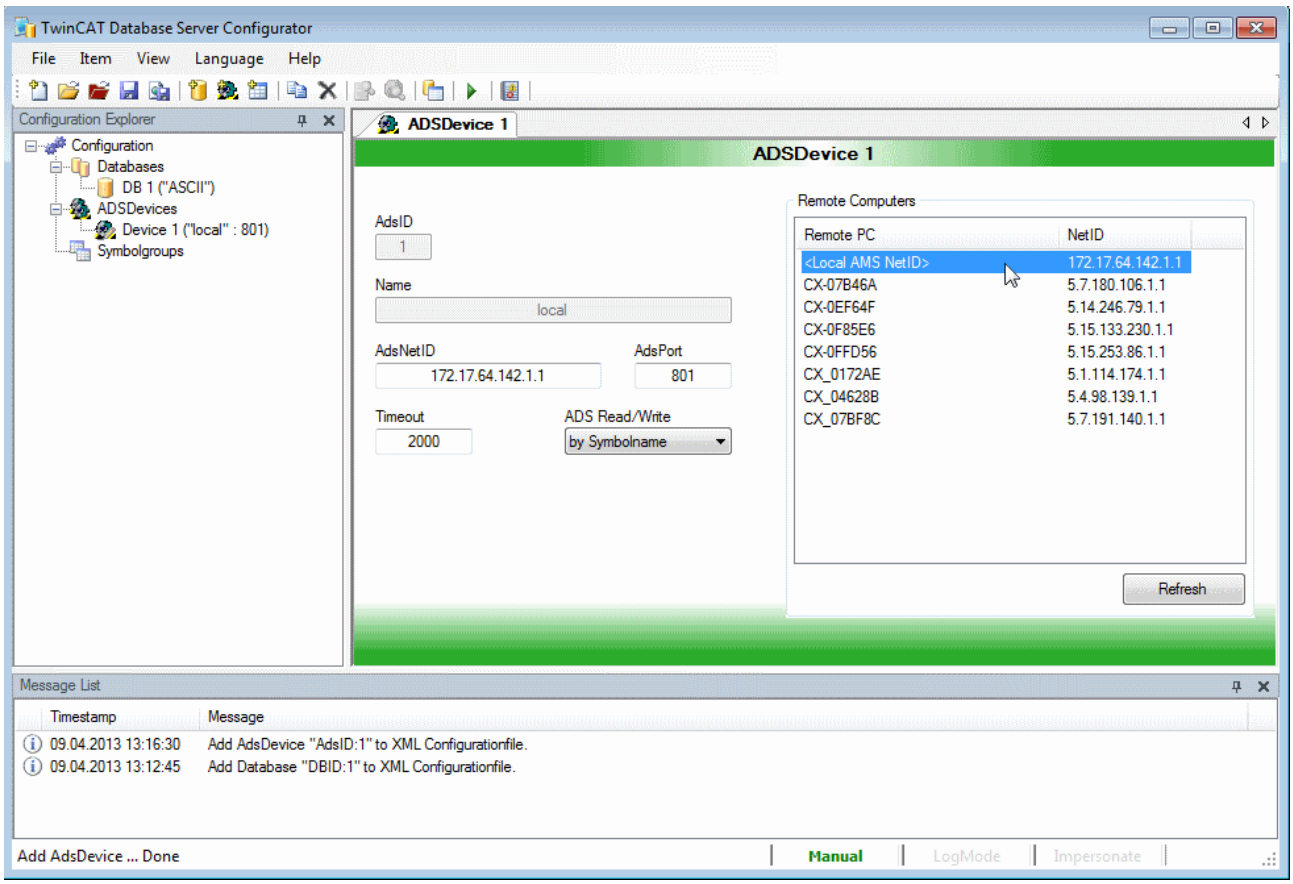
Choose the DBType ASCII. If you only will log alphanumeric data types and Boolean choose "Double" for DBValueType. Otherwise select "Bytes" for the log of structures and strings too.



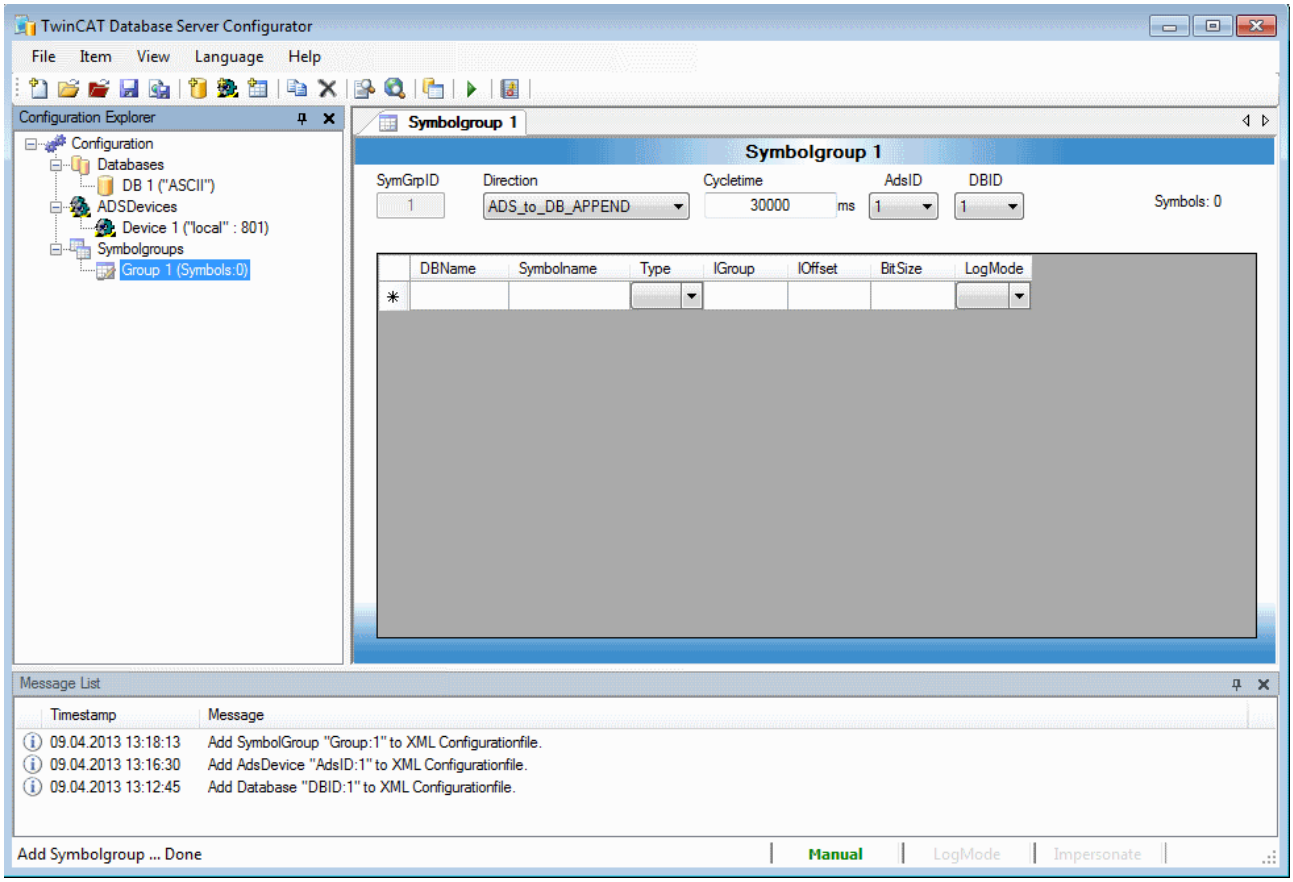
For an ASCII database you must indicate only the database URL, like it is shown in the next picture. You still need the Database ID (DBID) later on and also the AdsID which you can see on the next page.



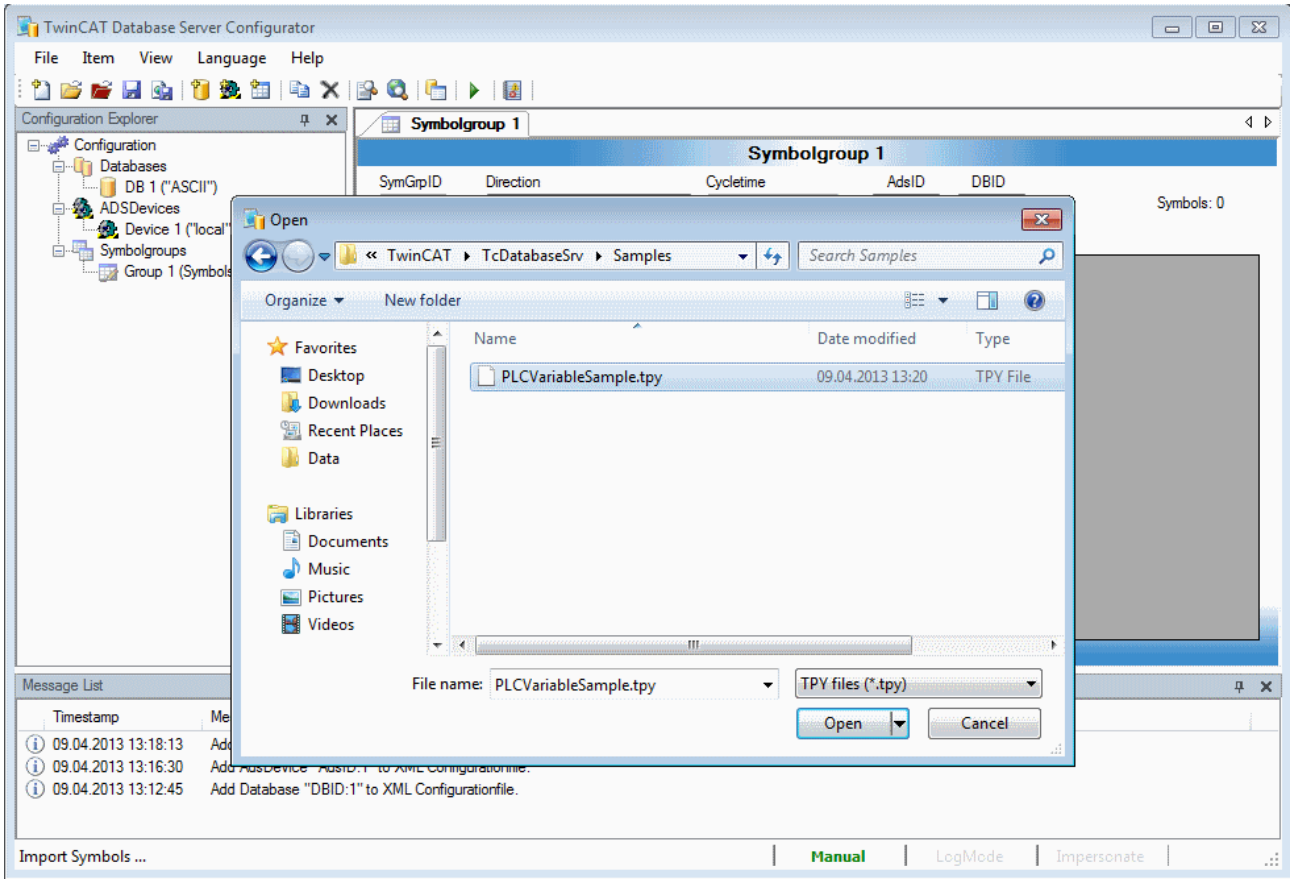
Add an AdsDevice and insert your AdsNetID, also your AdsPort. If you use the local system, you don’t need to type your NetID.

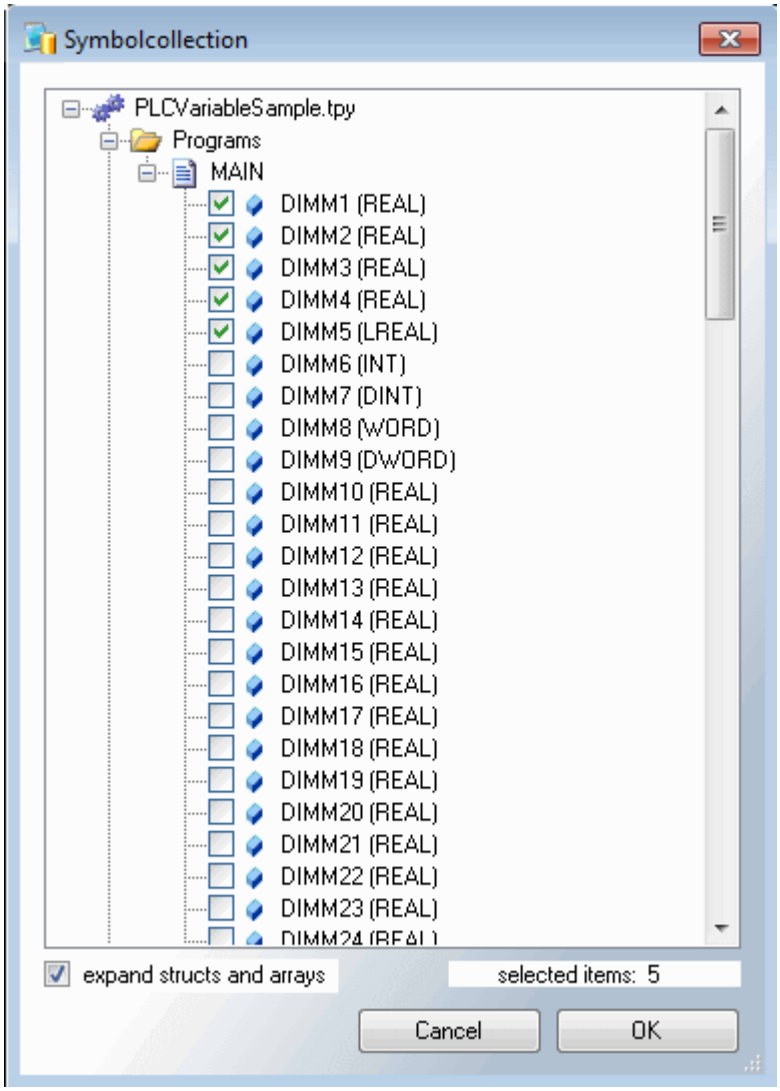


If you create a configuration for a remote system, you have the possibility to choose your target in the table on the right side. Now you can insert a Symbolgroup with variables from your PLC project.

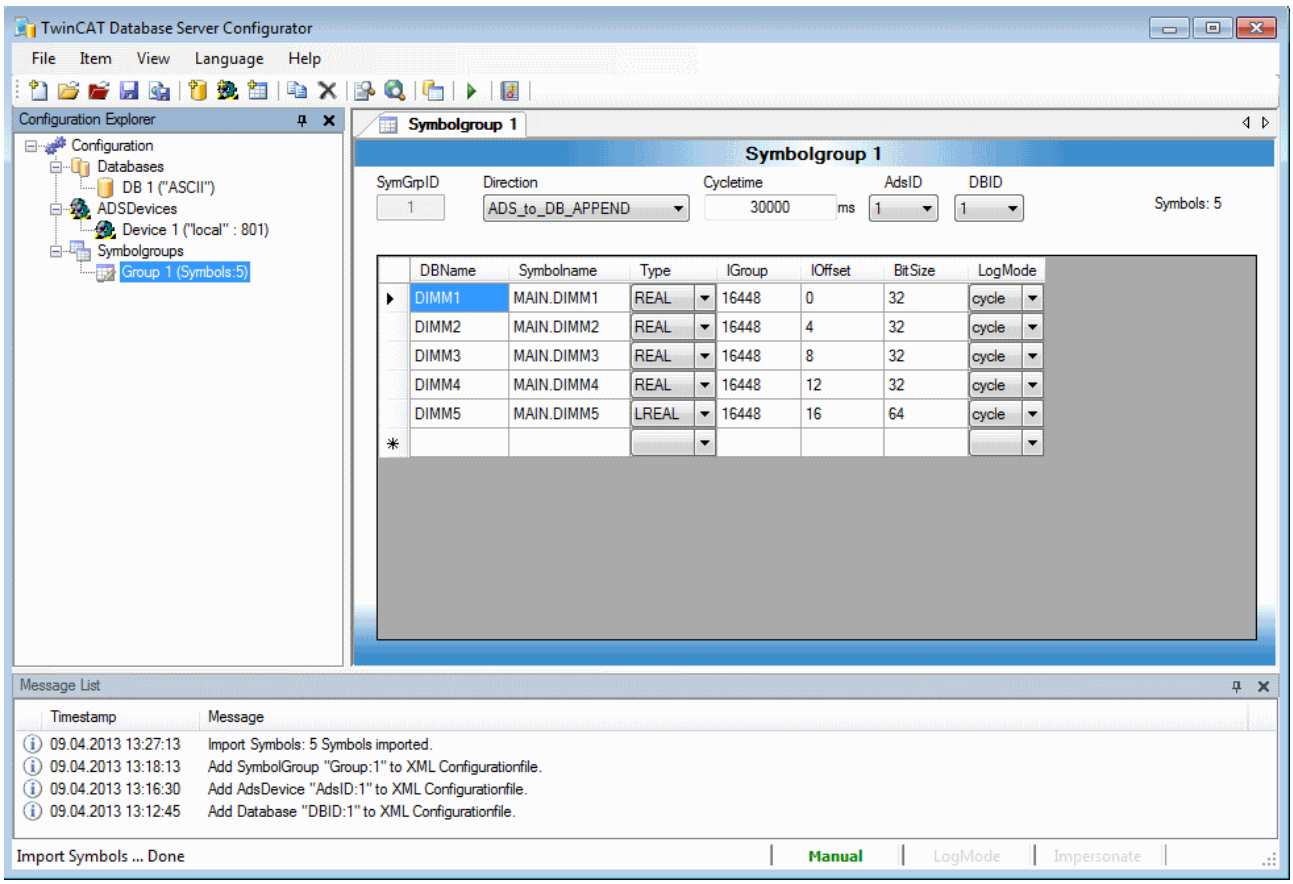


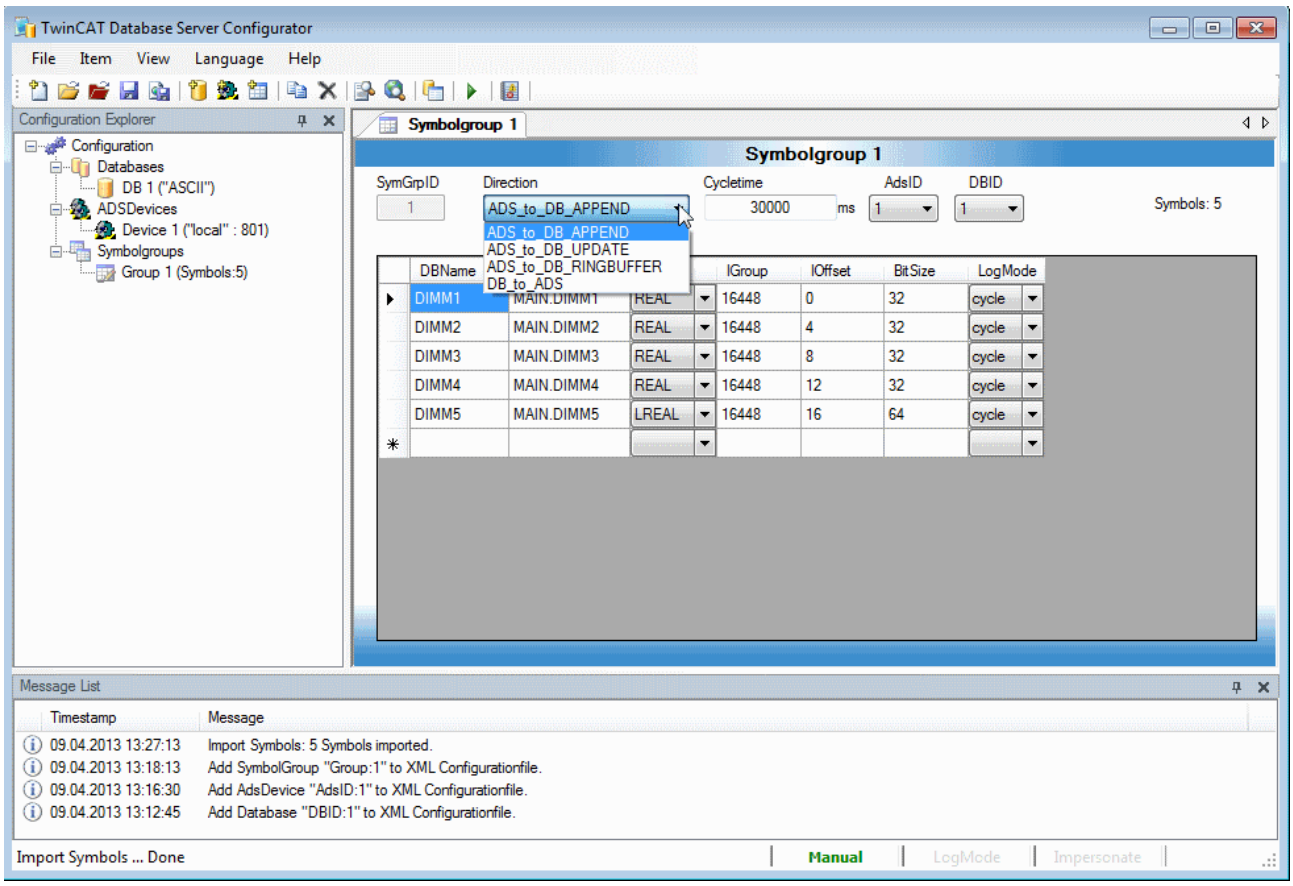
Before you configure your Symbol group, you have to “Build” your PLC project. Open TwinCAT PLC Control with PLCVariableSample.pro file and then Project -> Build. This is important for the generation of the tpy file. You can search for variables about the “Import Symbols” button in the Database Editor.





For example, you choose the first five variables of the Symbol collection. You can change the Log Mode for each variable, like it is shown in the next picture. You must set the AdsID and the DBID, which you know from the sides before.





You have the possibility to choose the communication direction. In this case ADS_to_DB_APPEND is selected.

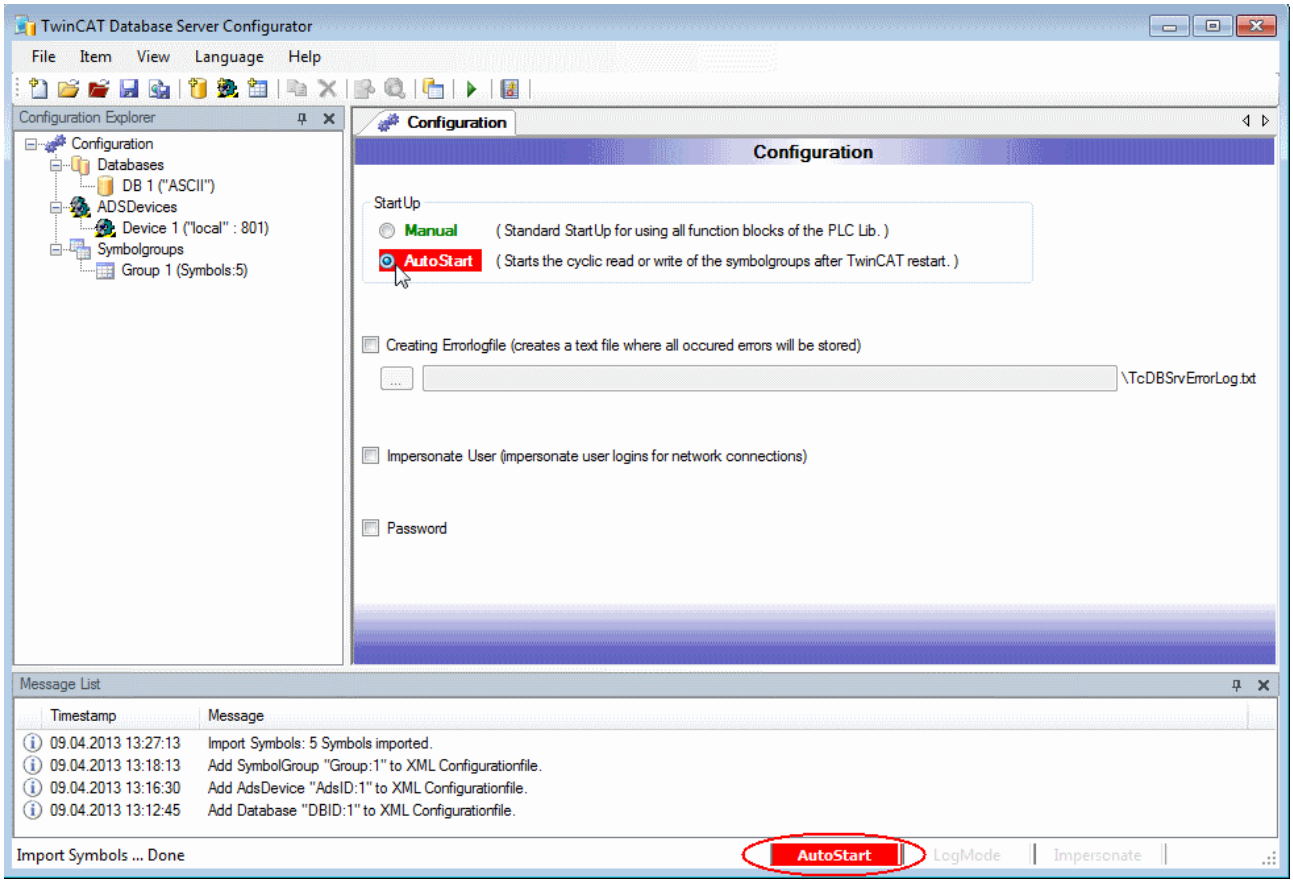
ADS_to_DB_APPEND:
This option appends the new data entries to the old in the database.

ADS_to_DB_UPDATE:
This option updates the available entries in the database.

ADS_to_DB_RINGBUFFER:
With this write mode you can limit the count or the age of datasets at databasetables.

DB_to_ADS:
This communication direction describes the reading of values from the database into the PLC.

If the Database Server should be used without the PLC (only logging from PLC variables), the option AutoStart must be set. So the Database Server immediately starts creating the connections to the declared database and ADS-devices after a TwinCAT start. Therefore you have to create a Bootproject from PLCVariableSample.pro. Finally you have to save your DB configuration in the Editor under C:\TwinCAT\Boot.



4. Configuration test

Go to PLC Control and start the PLC, after that create a Bootproject and make a TwinCAT restart. Look for the test database “TestDB” under C:\TwinCAT\TcDatabaseSrv\Samples.

This was a small introduction for the Beckhoff Database Server. Have fun with it. Questions? Then write me an email! For professionals follows a small instruction for the configuration of the Database Server from the PLC.

5. DataBaseServer Configuration from the PLC

Task:
Create a Microsoft Compact SQL database from the PLC Control, add a database connection and insert a new table in your database. Finally use the write function block with the ring buffer mode “RingBuffer_Count” to write 100 times a variable into the table of your database.

At first insert a R_TRIG function block to start your program with a rising edge. After that create a state machine for the call of your database function blocks, like it is shown in the picture below.

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

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

Now declare all other variables that you need. Especially the structure of the table. You can find a description how to do this for Microsoft Compact SQL in the Information System [▶ 40]:

An AutoID is generated in the “ID” column. The value in this column is always increased by 1. The “Timestamp” column stores the time at which the data record was saved. And the name of the variable is stored in the third column. Under “Value” you find the values of the variables.

You can take the remaining variables from the next picture.

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

The next steps.

FB_DBCreate:

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

In state number one you have to insert the FB_DBCreate. The path of the database must be given to this function block. But you must provide that this database does not exist yet. Otherwise, there is an error message. In state 11 you must wait until the function block is not busy.

FB_DBConnectionAdd:

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

In this function block you have to set the DBType and the DBValueType (Bytes or Double). And you already have to give your table a name. The other settings for Compact SQL databases are in the Information System [▶ 35]

FB_DBTableCreate:

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

One of the inputs of this function block is “hDBID”. You give over the value which you get from the FB_DBConnectionAdd. Furthermore you give over the size and the address of your table structure.

FB_DBWrite:

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

With the FB_DBWrite you can write the current values of the variable “iLogVariable” into the database all the time. But there will be stored only 100 values of the variable, if you set the nRingBufferCount to 100.

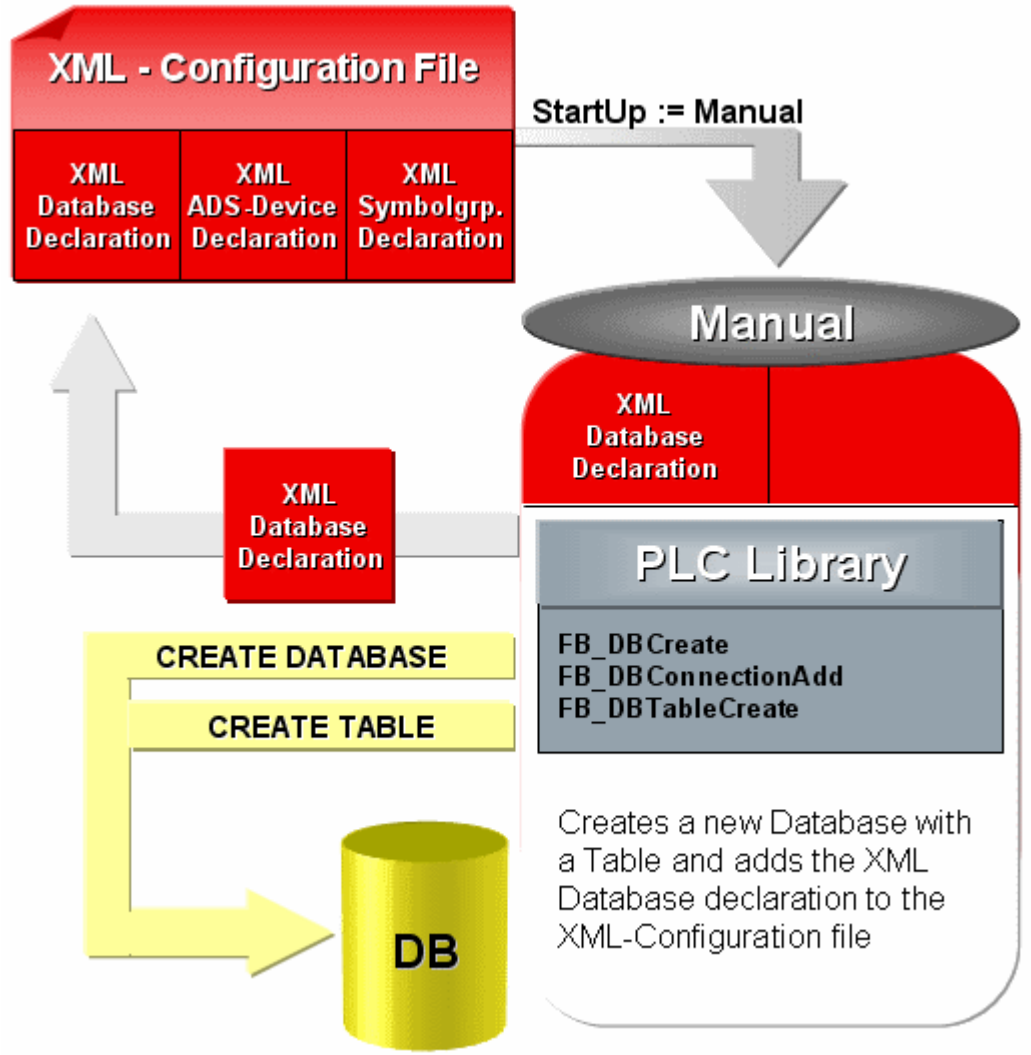
At the end of the program code, you can see the variable which is increasing by1 every PLC cycle.

Important: If you test the sample program take care that the database does not exist!

8.2 Create a Database

Download "Example generation of a database" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407901963/.zip

In this example the generation of a database out of the PLC will be shown.
In addition a table will be added and the generated database in the XML-configuration data declared.



Used database type	MS Access
Compatible database type	MS SQL, MS Compact SQL, MS Access
Used function blocks	FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib","TcStandard.lib"
Download data list	FB_DBCreate_Sample.pro

To use this example, you need to pass over the NetID of the ADS device (on which the TwinCAT Database Server is installed) to the input sNetID. And to the generated database a table named "myTable" will be added. This will have the following table structure:

Column name	Data type	Property
ID	bigint	IDENTITY(1,1)
Timestamp	datetime	
Name	ntext	
Value	float	

This table structure will be generated with the following array:

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

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_BIGIN
T),
                    (sColumnName:='Timestamp',eColumnType:=EDBCOLUMN_DATETIME),
                    (sColumnName:='Name',eColumnType:=EDBCOLUMN_NTEXT),
                    (sColumnName:='Value',eColumnType:=EDBCOLUMN_FLOAT);

    nState:BYTE := 0;

END_VAR
```

PLC Program

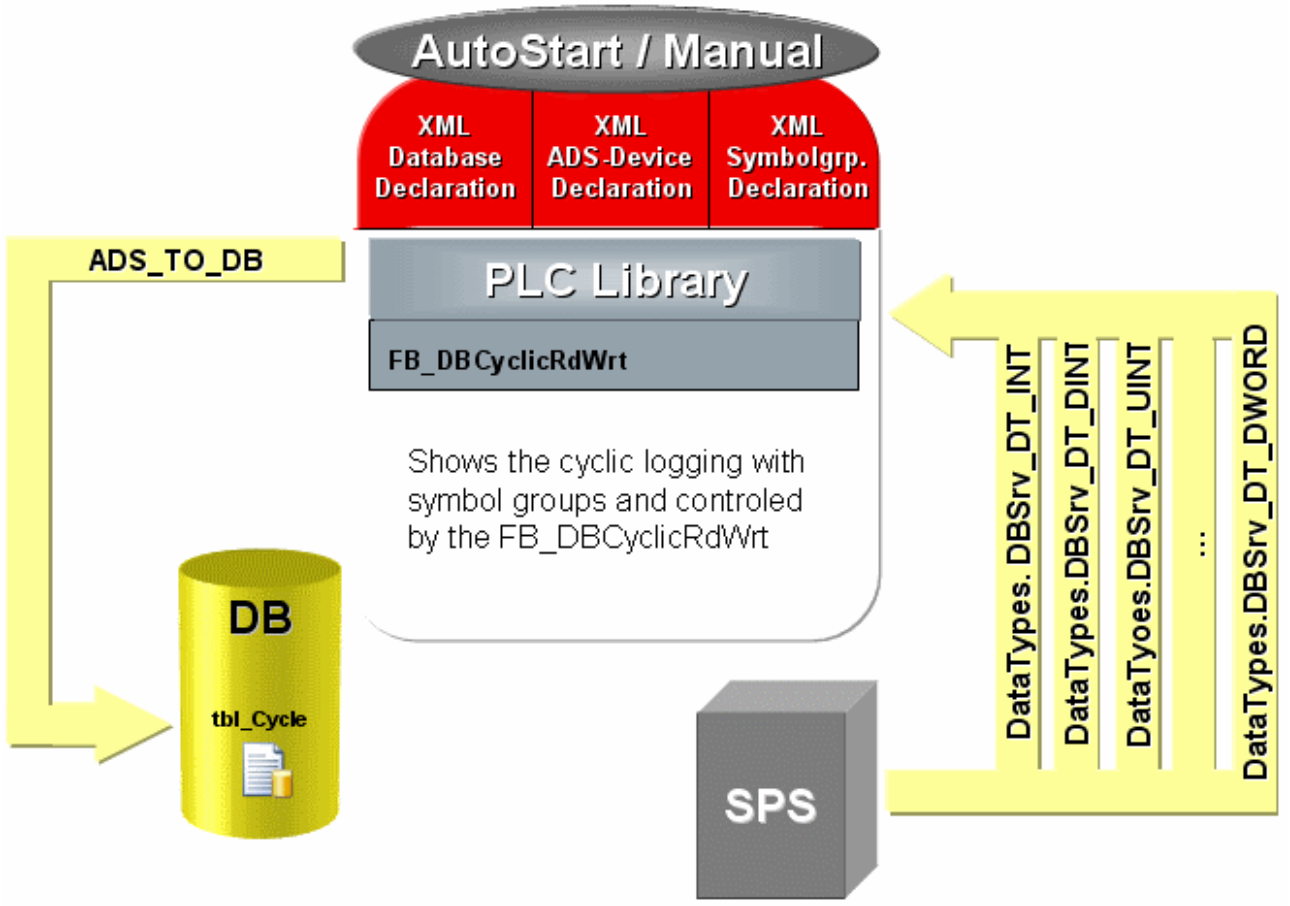
```
CASE nState OF
    0:
        (*To start this sample you have to set a rising edge to the variable bSTART*)
        R_TRIG1(CLK:=bSTART);
        IF R_TRIG1.Q THEN
            nState := 1;
            FB_FileDelete1(bExecute:=FALSE);
            FB_DBCreate1(bExecute:=FALSE);
            FB_DBConnectionAdd1(bExecute:=FALSE);
            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
```

8.3 Start / stop of cyclic logging with FB_DBCyclicRdWrt

Download "Example Start / Stop of the cyclic log" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407903371/.zip

In this example the start and stop of the cyclic log out of the PLC will be shown.



Used database type	MS Compact SQL
Compatible databse type	ASCII, MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird
Used function blocks	FB_DBCyclicRdWrt
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib","TcStandard.lib"
Download data list	FB_DBCyclicRdWrt_Sample.pro, CurrentConfigDataBase.xml, TestDB_Cyclic.sdf

In this example the cyclic log function will be started or stopped by toggling of the bStartStop variable.

At a positive edge at the bExecute input the cyclic log operation starts. At a negative edge this will be stoped.

Variable declaration (PRG DataTypes)

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

Structure ST_MyStruct

```
TYPE ST_MyStruct :
STRUCT
    iValue1 : INT;
    iValue2 : UINT;
```

```
    iValue3 : BOOL;
    iValue4 : REAL;
END_STRUCTEND_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);
```

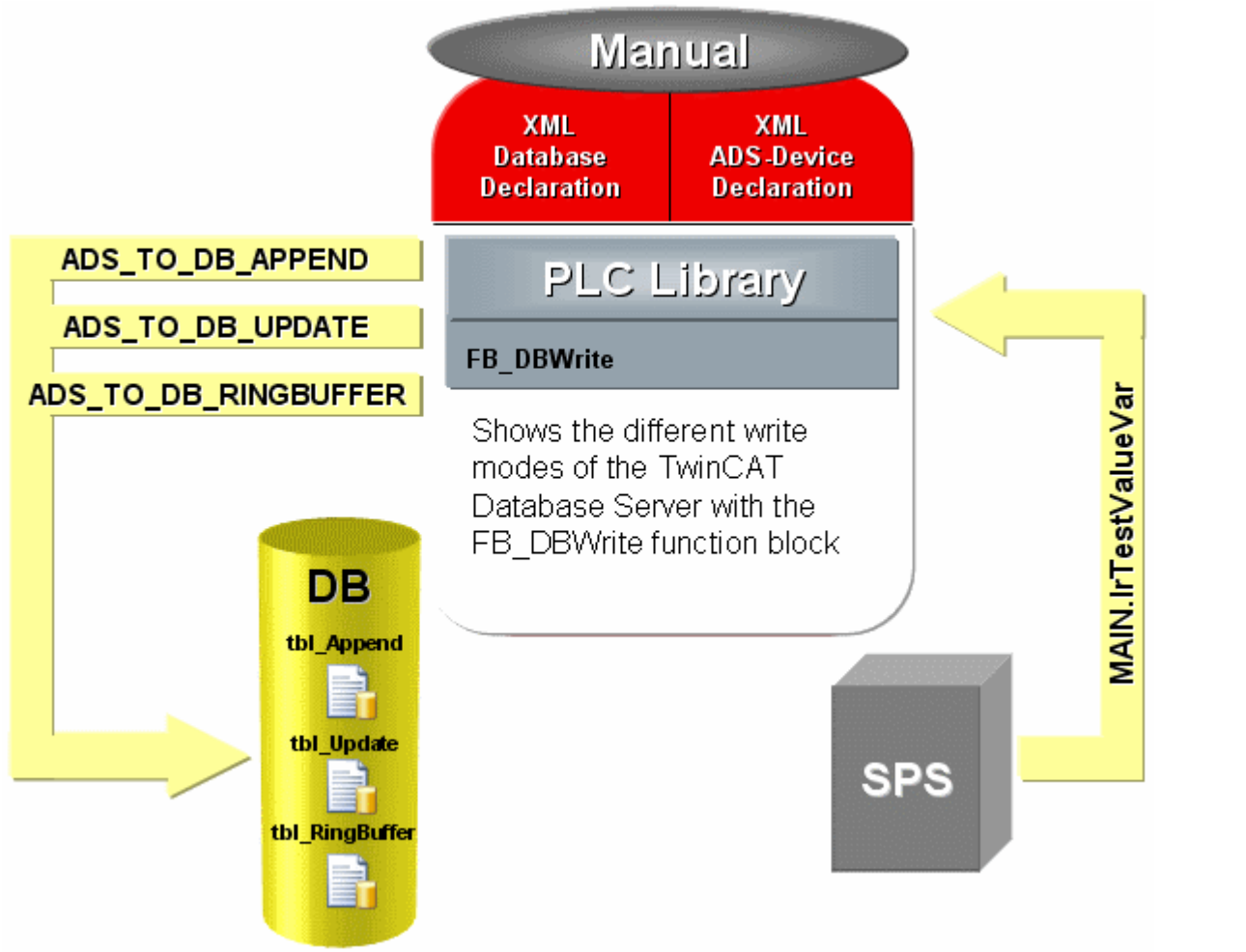
To use this example, you need to pass over the NetID of the ADS device (on which the TwinCAT Database Server is installed) to the input sNetID.
If you start the program and set the bCyclic variable to TRUE, all variables will be logged, that are declared in the symbol group of the XML-configuration data.

All the Microsoft SQL Compact databases that 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.

8.4 Logging of one PLC variable with FB_DBWrite

Download: "Example to log a PLC variable with FB_DBWrite" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407904779/.zip

In this example the log of a PLC variable out of the PLC into a database will be demonstrated. The functionalities of each write-modes are shown.



Used database type	MS SQL
Compatible database type	ASCII, MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird
Used function blocks	FB_DBWrite
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib","TcStandard.lib"
Download data list	FB_DBWrite_Sample.pro, CurrentConfigDataBase.xml, SQLQuery.sql

To use this example, you need to adjust the server name and the authentication in the XML-configuration data.

Furthermore, you need to keep in mind that no "TestDB" database is available before you carry out the SQLQuery.sql Script.

Configuration example:

With the help of the variable "eWriteMode" you can set with which write-mode it should be logged.
With a rising edge at the variable "bSTART" the write operation can be started.

Table:

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

Used table structure for each table

Column name	Data type	Zero allowed	Property
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
```

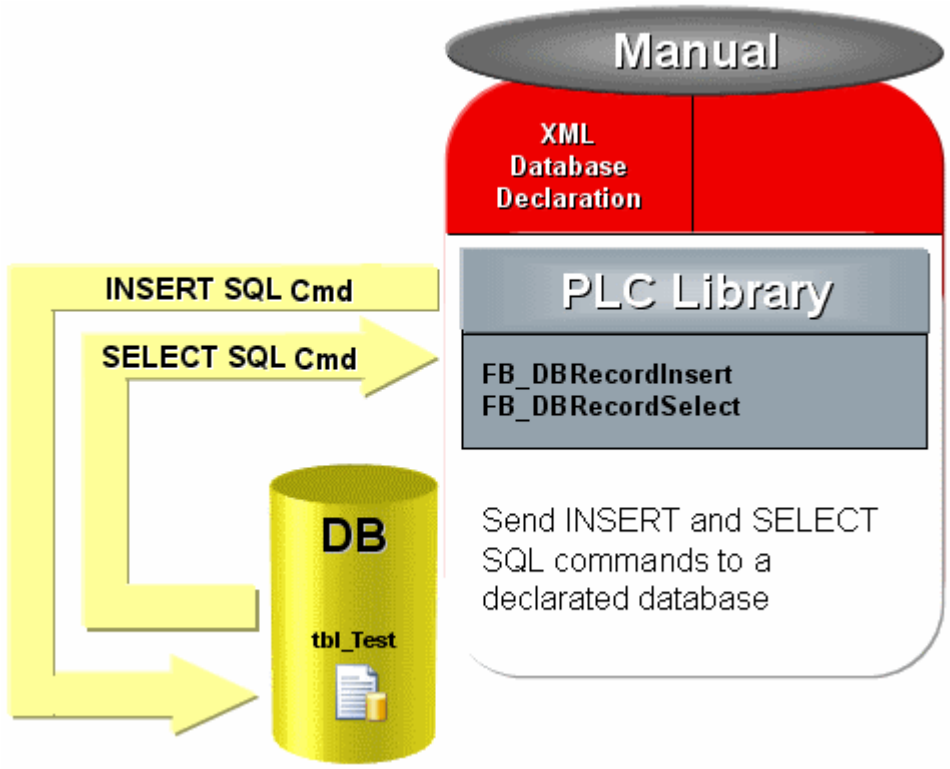
```
            FB_DBWrite_Append(bExecute := FALSE);
            eState := eIdle;
        END_IF(*Add a new record to the table tbl_Update if it not exist else the existing record wi
ll be updated*)
    eWriteUpdate :
        FB_DBWrite_Update(
            sNetID:= ,
            hDBID:= 2,
            hAdsID:= 1,
            sVarName:= 'MAIN.lrTestValueVar',
            nIGroup:= ,
            nIOffset:= ,
            nVarSize:= ,
            sVarType:= ,
            sDBVarName:= 'lrTestValueVar',
            eDBWriteMode:= eDBWriteMode_Update,
            tRingBufferTime:= ,
            nRingBufferCount:= ,
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy,
            bError=> bErr,
            nErrID=> bErrid,
            sSQLState=> stSqlstate);

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

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

All the Microsoft SQL Compact databases that 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.

8.5 Sample with FB_DBRecordInsert/FB_DBRecordSelect

Download "Example to log with the FB_DBRecordInsert" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407906187/.zip

In this example the log of several values into a database out of a PLC with the function block FB_DBRecordInsert will be shown.
Especially in this example several PLC variables will be logged in one data set. Furthermore a data set can be read out of the database with the function block FB_DBRecordSelect.



Used database type	MS Access
Compatible database types	MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird
Used function blocks	FB_DBRecordInsert, FB_DBRecordSelect
Integrated libraries	**"TcDatabase.lib"**, "TcSystem.lib","TcBase.lib","TcStandard.lib","TcUtilities.lib"
Download data list	FB_DBRecodInsertSelectSample.pro, CurrentConfigDataBase.xml, TestDB_Access.mdb

To use this example, you need to declare the access database "Sample7.mdb" in the XML-configuration data.

By generating a positive edge at the variable "bStartstopInsert ", a data set with four PLC values and the timestamp will be attached in the database.

The following table structure will be written:

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

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

Struct ST_Record

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

PLC Program

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

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

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

```
            nErrId=> nErrid);

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

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

Screenshot tbl_Test:

tbl_Test : Tabelle

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

Datensatz: 7 von 7

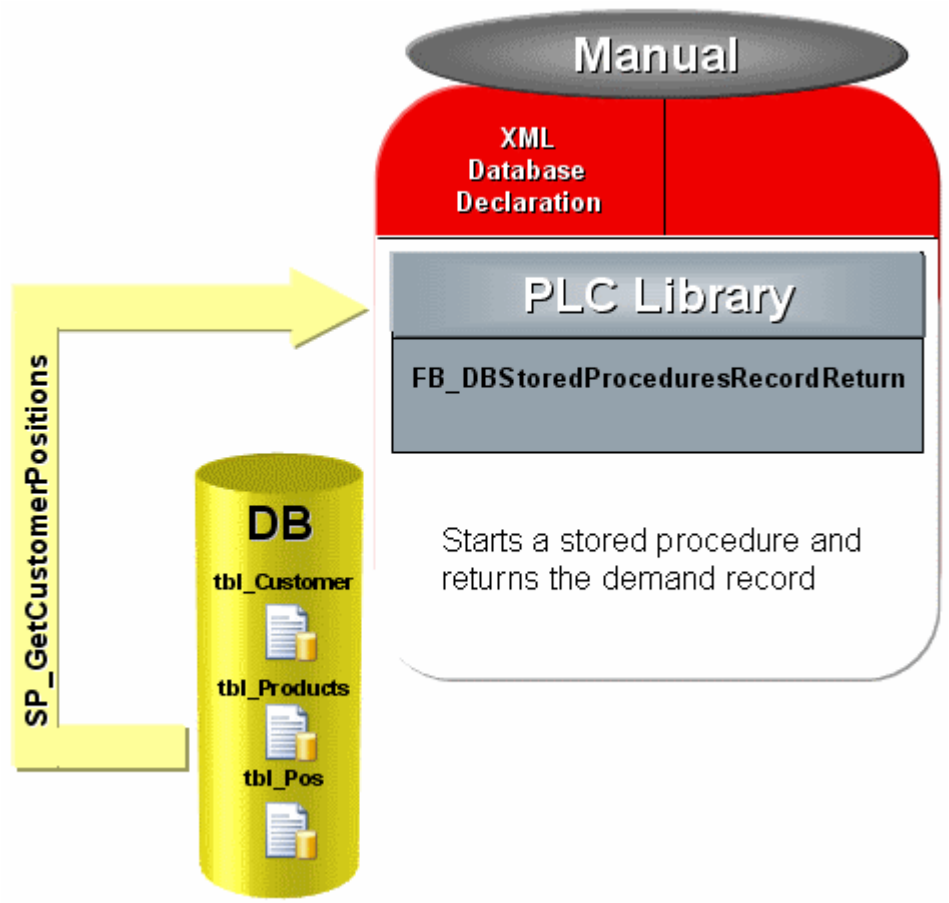
8.6 Stored Procedures with MS SQL

No "stored procedures" can be generated or configurated with the Database Server.

Stored procedures can be carried out with the function blocks FB_DBStoredProcedures and FB_DBStoredProceduresRecordReturn from the version 1.0.13 on.

With the help of these function blocks parameters can be declared as INPUT, OUTPUT or INOUT and can be passed over to the stored procedures. So complex SQL-commands can be pre-programmed at the database server and only need to be triggered by the TwinCAT Database Server.

Download "Example with stored procedures" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407907595/.zip



Used database type	MS SQL (MS SQL Server 2008)
Compatible database types	MS SQL
Used function blocks	FB_DBStoredProceduresRecordReturn
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib","TcStandard.lib"
Download data list	FB_DBStoredProcedures_Sample.pro, CurrentConfigDataBase.xml, SQLQuery2.sql

The following example shows the call of a stored procedure with an input parameter and a return data set for a customer and production database sample. The procedure was generated at a Microsoft SQL Server 2008.

Code of the 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_Produ
cts.Product, tbl_Products.Info, tbl_Pos.Timestamp FROM
        tbl_Pos JOIN tbl_Customer ON tbl_Pos.CustomerNum = tbl_Customer.ID
        JOIN tbl_Products ON tbl_Pos.ProductNum = tbl_Products.SerNum
    WHERE
        tbl_Pos.CustomerNum = @Customer_ID;
END
```

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;
```

```
    nRecordIndex: UDINT;
    stRecord: ST_Record;
    nRecs: UDINT;

    FB_DBStoredProceduresRecordReturn1: FB_DBStoredProceduresRecordReturn;

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

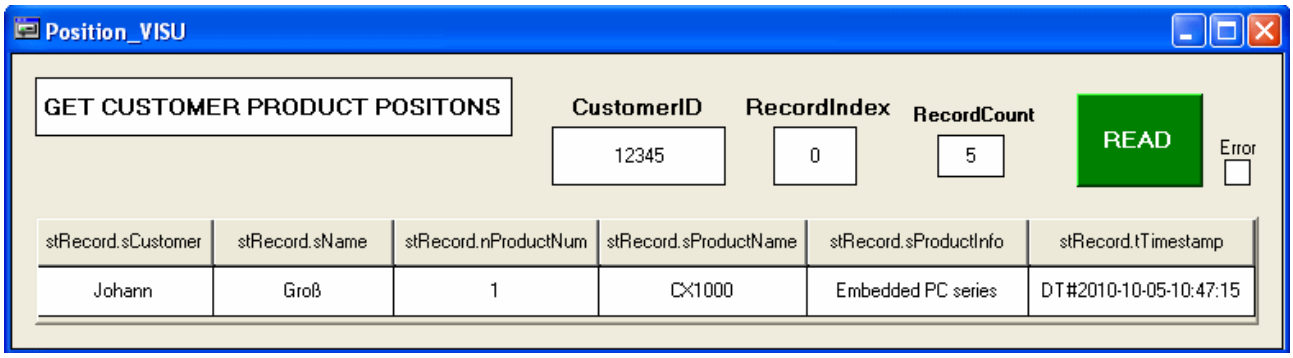
Data set structure in the PLC (ST_Record)

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

PLC Program

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

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

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

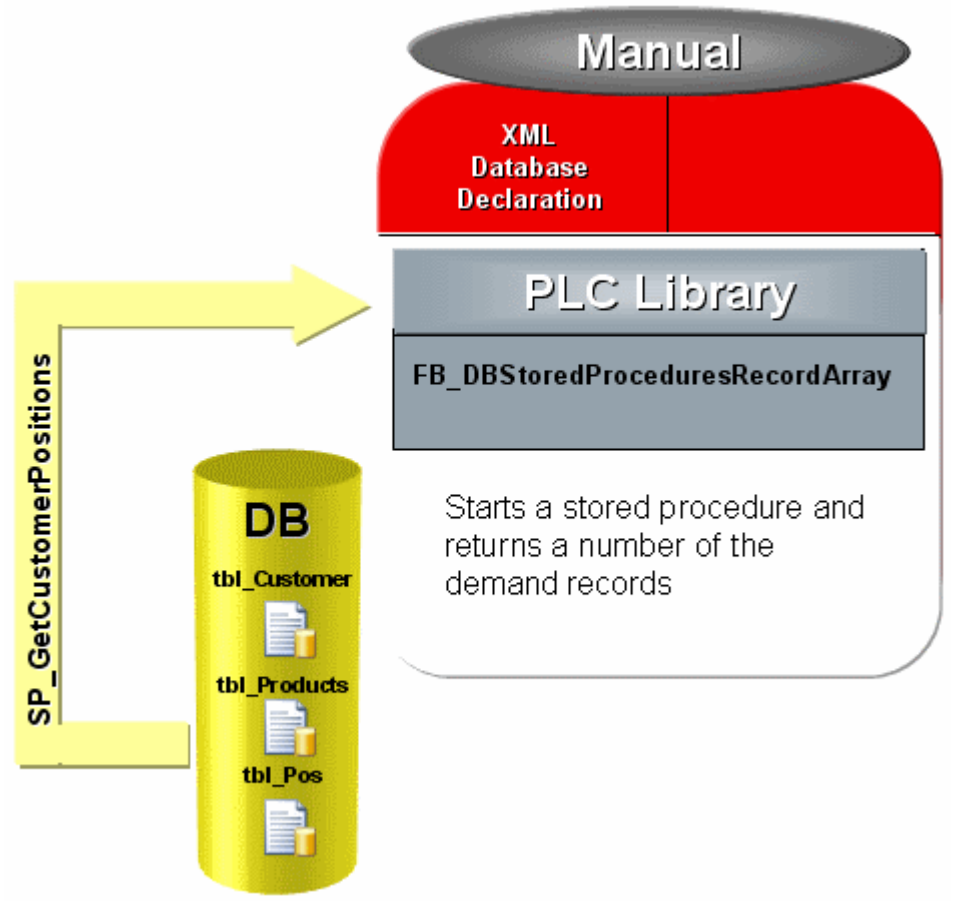
Visualization



8.7 Stored Procedures with FB_DBStoredProcedureRecordArray

With the help of the function block FB_DBStoredProceduresRecordArray parameters can be declared as INPUT, OUTPUT or INOUT and can be passed over to the stored procedures. So complex SQL-commands can be pre-programmed at the database server and only need to be triggered by the TwinCAT Database Server. The difference to the FB_DBStoredProceduresRecordReturn function block is, you get a number of records with only one function call.

Download "Example with stored procedures" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407909003/.zip



Used database type	MS SQL (MS SQL Server 2008)
Compatible database types	MS SQL
Used function blocks	FB_DBStoredProceduresRecordArray
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib","TcStandard.lib"

Download data list	FB_DBStoredProceduresRecordArray_Sample.pro, CurrentConfigDataBase.xml, SQLQuery2.sql

The following example shows the call of a stored procedure with an input parameter and a return data set for a customer and production database sample. The procedure was generated at a Microsoft SQL Server 2008.

Code of the 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_Produ
cts.Product, tbl_Products.Info, tbl_Pos.Timestamp FROM
        tbl_Pos JOIN tbl_Customer ON tbl_Pos.CustomerNum = tbl_Customer.ID
        JOIN tbl_Products ON tbl_Pos.ProductNum = tbl_Products.SerNum
    WHERE
        tbl_Pos.CustomerNum = @Customer_ID;
END
```

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
```

Data set structure in the PLC (ST_Record)

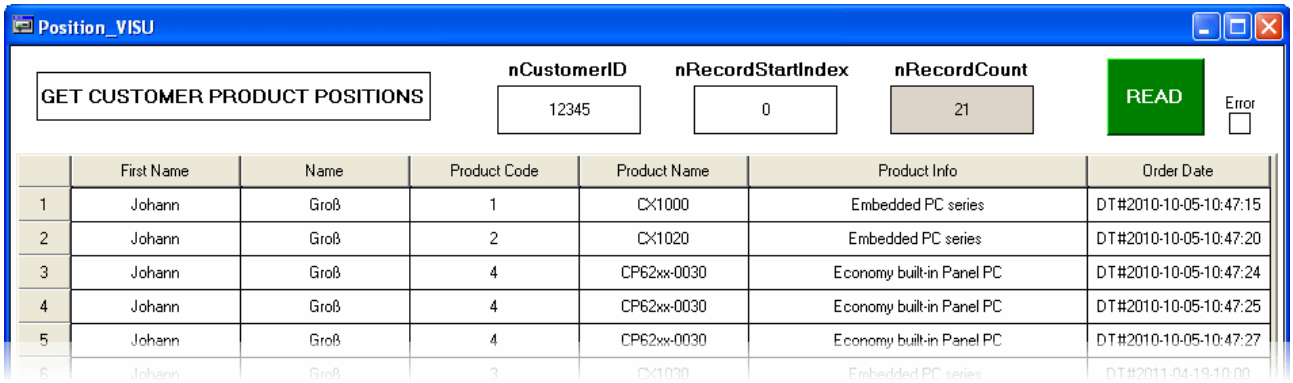
```
TYPE ST_Record :
STRUCT
    nID : T_ULARGE_INTEGER;
    sCustomer : STRING(50);
    sName : STRING(50);
    nProductNum : DINT;
    sProductName : STRING(50);
    sProductInfo : T_MaxString;
    tTimestamp : DT;
END_STRUCT
END_TYPE
```

PLC Program

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

        nState := 2;
```

```
2:(*Start the stored procedure "SP_GetCustomerPosition"*)
    FB_DBStoredProceduresRecordArray1(
        sNetID:= ,
        hDBID:= 1,
        sProcedureName:= 'SP_GetCustomerPositions',
        cbParameterList:= SIZEOF(arrParaList),
        pParameterList:= ADR(arrParaList),
        nStartIndex:= nRecordStartIndex,
        nRecordCount:= 25,
        cbRecordArraySize:= SIZEOF(stRecordArr),
        pDestAddr:= ADR(stRecordArr),
        bExecute:= TRUE,
        tTimeout:= T#15s,
        bBusy=> bBusy,
        bError=> bErr,
        nErrID=> nErrid,
        sSQLState=> stSqlstate,
        nRecords=> nRecs);

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

Visualization

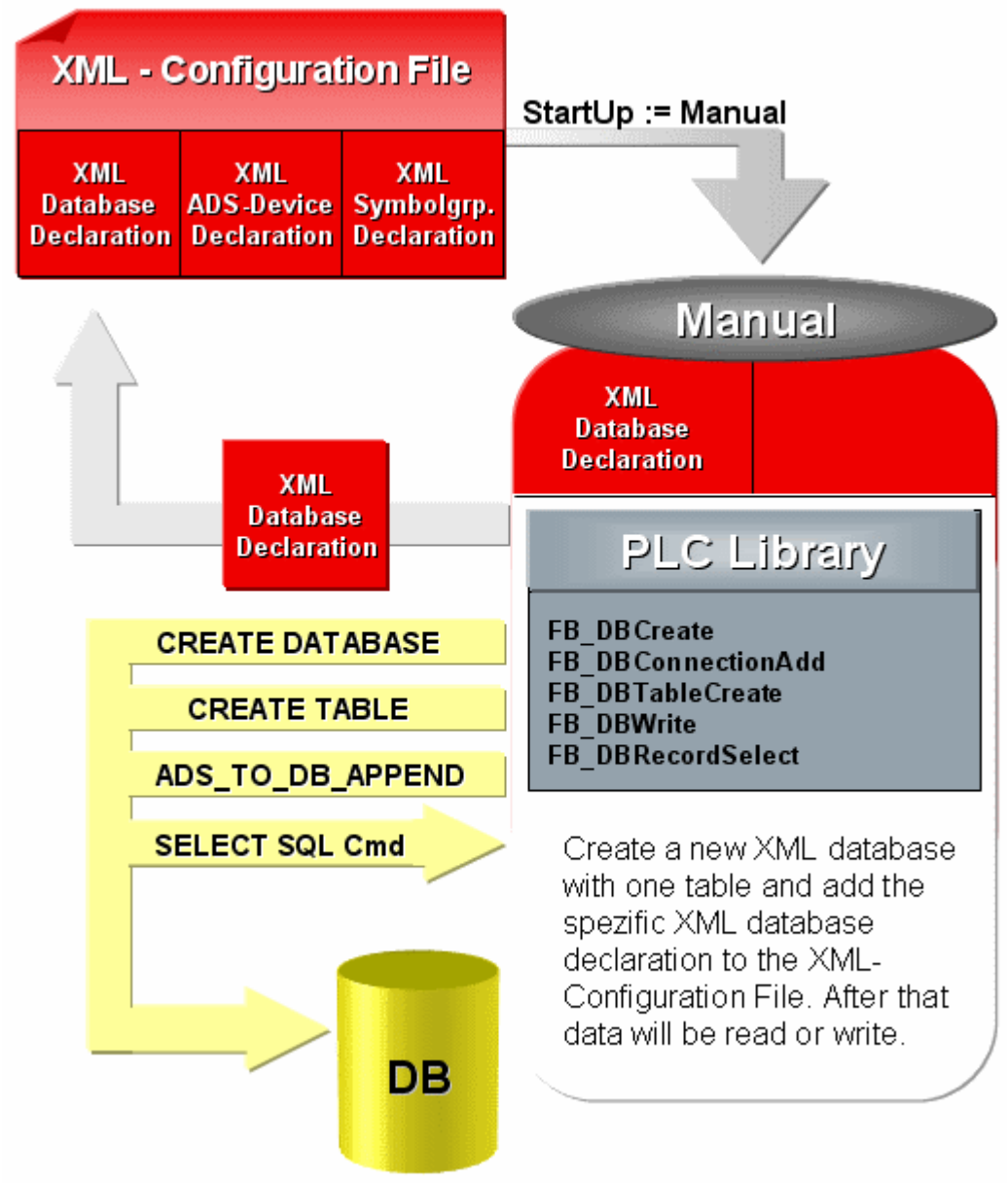


8.8 Use XML as Database

It is possible with TF6420 Database Server to use XML files as database. The TwinCAT3 Database Server supports all known function blocks for reading and writing to a database down to the function blocks for stored procedures. SQL queries which will be send with the function blocks FB_DBRecordInsert or FB_DBRecordSelect are interpreted of the TwinCAT3 Database Server and adequate use to the XML file.

This sample demonstrate how to create an XML database, fill with the function block FB_DBWrite and read the items with an SQL-SELECT query with the FB_DBRecordSelect of the created XML file.

Download "Sample to use mit XML as Database" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407910411/.zip



Used database type	XML
Compatible database type	MS SQL, MS Compact SQL, MS Access, XML
Used function blocks	FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate, FB_DBWrite, FB_DBRecordSelect
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib", "TcStandard.lib", "TcUtilities.lib"
Download data list	XML_DatabaseType.pro

MAIN Program

```
PROGRAMMAINVAR
    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
```

```
CASEnState 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=> );

        IFNOT 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=> );

        IFNOT 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);

        IFNOT bBusy_CreateDB ANDNOT 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);

           IFNOT bBusy_ConnAdd ANDNOT 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);

        IFNOTbBusy_CreateTable ANDNOT 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);

           IFNOT bBusy_WriteDB ANDNOT bErr THEN
            FB_DBWrite1(bExecute := FALSE);
            nCounter := nCounter + 1;
            IFnCounter = 5 THEN
                nState := 6;
            END_IFEND_IF
    6:
        (*The FB_DBRecordSelect select one record of the database table "myTable""*)
```

```
        FB_DBRecordSelect1(
            sNetID:= ,
            hDBID:= nDBid,
            sSelectCmd:= 'SELECT * FROM myTable WHERE Name = $'rTestValue$'',
            nRecordIndex:= 0,
            cbRecordSize:= SIZEOF(stRecord),
            pDestAddr:= ADR(stRecord),
            bExecute:= TRUE,
            tTimeout:= T#15s,
            bBusy=> bBusy_SelectRecord,
            bError=> bErr,
            nErrID=> nErrid,
            sSQLState=> stSQLState,
            nRecords=> nRecs);

        IFNOT bBusy_SelectRecord ANDNOT bErr THEN
            nState := 0;
        END_IFEND_CASE
```

Start the sample with a rising edge at the toggle variable bSTART..

Following files will be created:

XMLTestDB.xml (XML database file)

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

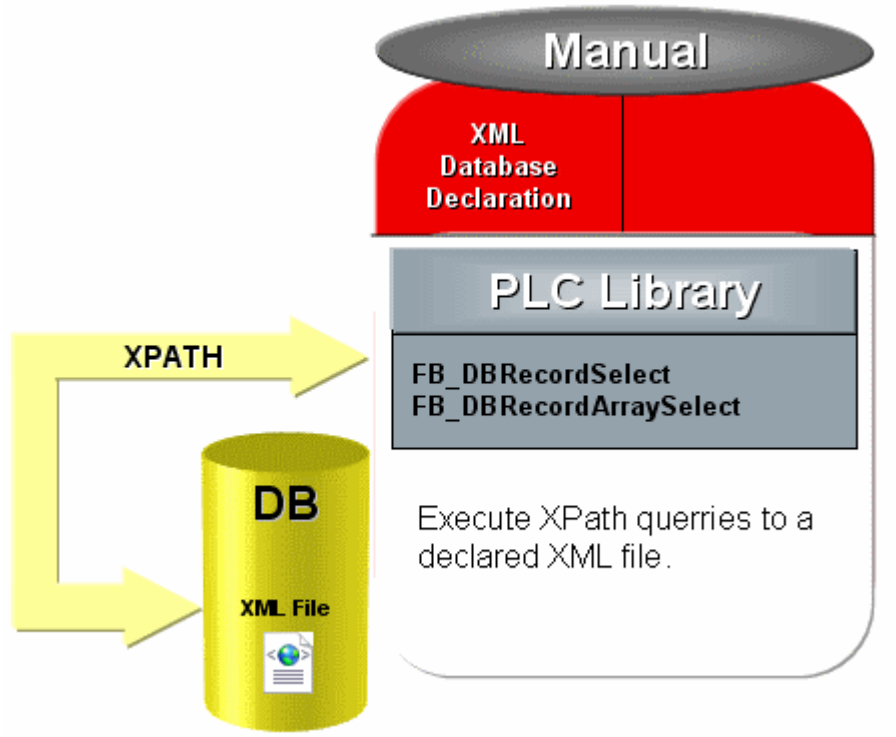
XMLTestDB.xsd (XML Schema)

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

8.9 XML XPath Sample for Visualisation

With the help of the function block FB_DBRecordSelect it is possible to send XPath querries to select XML-Tags of various XML files. This sample shows, how to select entries of dynamic textlists (XML-file) for the TwinCAT PLC Visualisation with the TF6420 Database Server.

Download "Beispiel mit XPath und TargetVisu" https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407911819/.zip



Used database type	XML
Compatible database type	XML
Used function blocks	FB_DBRecordSelect
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib","TcStandard.lib","TcUtilities.lib"
Download Data list	XPath_GetText.pro, CurrentConfigDatabase.xml, VisuTest.xml

Dynamic text list for TwinCAT PLC Visualization

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

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

Function block "FB_GetText" (to read the XML-Tags)

```
FUNCTION_BLOCKFB_GetText
VAR_INPUT
    dwID: DWORD;
    stPrefix: T_MaxString;
    stLanguage : T_MaxString;
    bExecute : BOOL;
END_VAR

VAR_OUTPUT
    bBusy : BOOL;
    bError : BOOL;
    nResultLength : INT;
    stResult : STRING(256);
END_VAR

VAR
    R_TRIG1: R_TRIG;
    state: BYTE;
    FB_DBRecordSelect1: FB_DBRecordSelect;
    FB_FormatString1: FB_FormatString;
END_VAR
```

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

    FB_DBRecordSelect1(
        sNetID:= ,
        hDBID:= 1,
        nRecordIndex:= 0,
        cbRecordSize:= SIZEOF(stResult),
```

```
        pDestAddr:= ADR(stResult),
        bExecute:= TRUE,
        tTimeout:= T#10s);

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

MAIN Program

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

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

The function block FB_GetText will be execute with a rising edge at the input variable bExecute. The received text contains the output variable stResult. The length of the text will be returned at the output variable nResultLength.

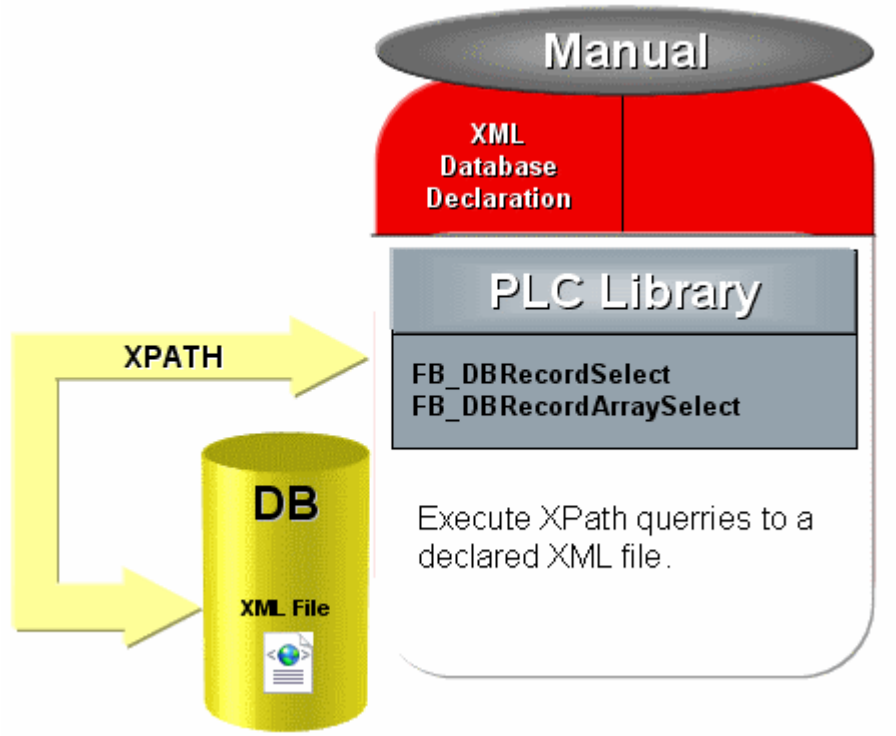
8.10 XML XPath Sample with XML Schema

With the help of the function block FB_DBRecordSelect or FB_DBRecordArraySelect it is possible to execute XPath querries select XML-Tags, XML-Subtags or XML-Attributes of any XML files. If an properly XML schema is available, the entries of the tags or attributes will be converted to the right data types.

Xou can find further information about XML-Schemes here: http://www.edition-w3.de/TR/2001/REC-xmlschema-0-20010502/

This sample demonstrate the reading of two different sub tags from a XML file with its appropriate schema.

Download https://infosys.beckhoff.com/content/1033/tcdbserver/Resources/11407913227/.zip



Used database type	XML
Compatible database type	XML
Used function blocks	FB_DBRecordSelect
Integrated libraries	"**TcDatabase.lib**", "TcSystem.lib","TcBase.lib", "TcStandard.lib", "TcUtilities.lib"
Download data list	XPath_XMLSubTag.pro, 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>
```

Appropriate 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_STRUCTEND_TYPE
```

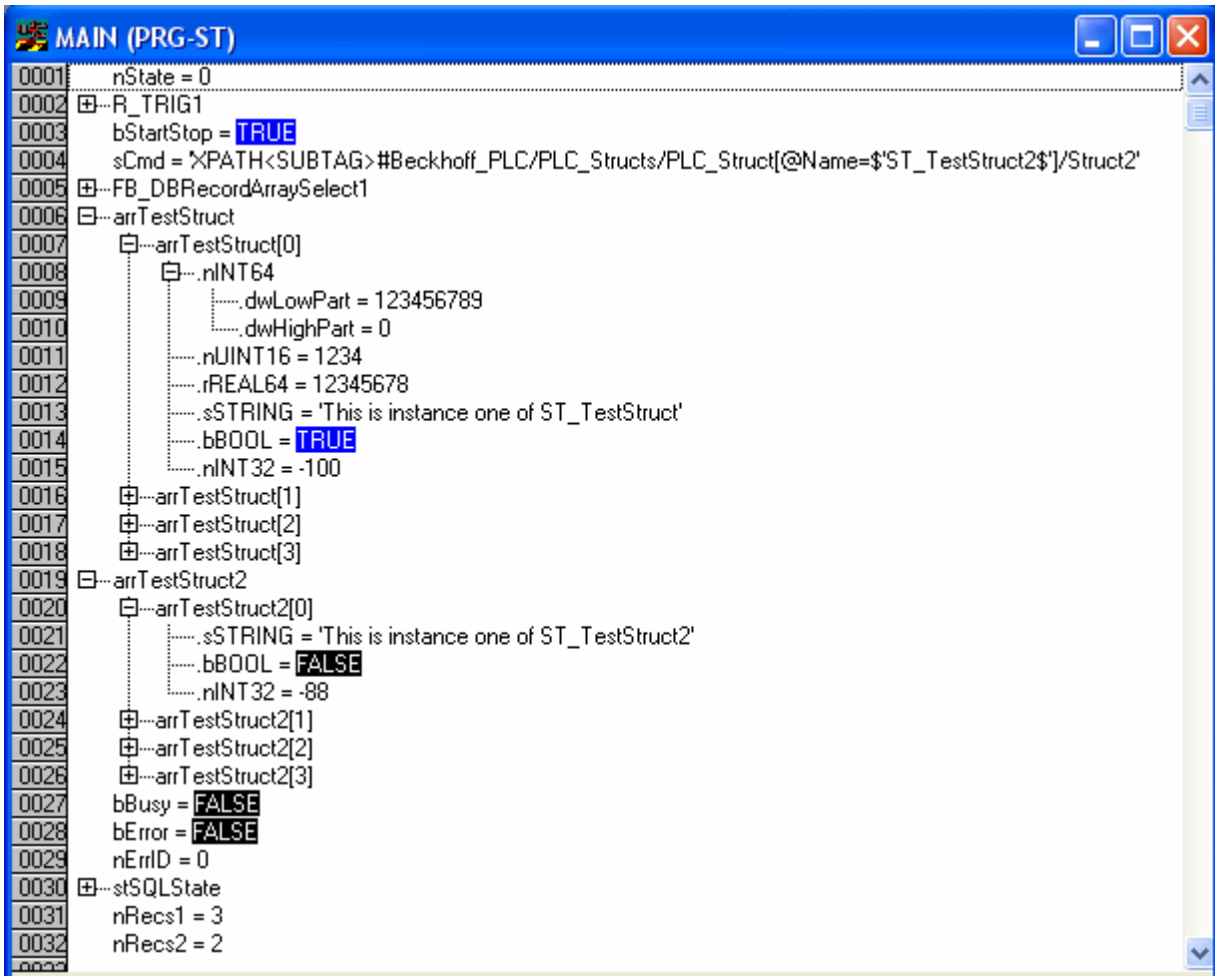
MAIN Program

```
PROGRAMMAIN
VAR
    nState: BYTE;

    R_TRIG1: R_TRIG;
    bStartStop: BOOL;

    sCmd: T_MaxString;

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

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

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

```
R_TRIG1(CLK:=bStartStop);
IF R_TRIG1.Q THEN
    FB_DBRecordArraySelect1(bExecute:=FALSE);
    nState := 1;
END_IF

CASE nState OF
 0:(*Idle*)
    ;
 1:
    sCmd:='XPATH<SUBTAG>#/Beckhoff_PLC/PLC_Structs/PLC_Struct[@Name=$'ST_TestStruct$']/Struct';

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

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

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

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

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

Start the reading with a rising edge at the toggle variable bStartStop.



9 Appendix

9.1 Errorcodes

9.1.1 ADS Return Codes

Error codes: 0x000... [▶ 132], 0x500... [▶ 132], 0x700... [▶ 132], 0x1000... [▶ 132], 0x274C... [▶ 132]

Global Error Codes

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

Router Error Codes

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

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

General ADS Error Codes

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

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

RTime Error Codes

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

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

TCP Winsock Error Codes

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

9.1.2 Internal Errorcodes of the TwinCAT Database Server

Code (Hex)	Code (Dez)	Description
0x0001 + ADS-FehlerCode	65537 - 131071	ADS Errorcode of the declared ADS-Device

Code (Hex)	Code (Dez)	Description
0x00020001	131073	Microsoft SQL Compact Database (Errorcode)
0x00040001	262145	Microsoft SQL Database (Errorcode)
0x00080001	524289	Microsoft Access Database (Errorcode)
0x00100001	1048577	MySQL Database (Errorcode)
0x00200001	2097153	Oracle Database (Errorcode)
0x00400001	4194305	DB2 Database (Errorcode)
0x00800001	8388609	PostgreSQL Database (Errorcode)
0x01000001	16777217	Interbase/Firebird Database (Errorcode)
0x02000001	33554433	TwinCAT Database Server Errorcode
0x04000001	67108865	XML Database (Errorcode)
0x08000001	134217729	ASCII Database (Errorcode)

If an error occurs at executing an SQL statement, one of the declared errorcodes from the top of this site will be displayed at the output "nErrID". The specified errorcode of the database will be displayed at the "sSQLState" Output of the function block. The output "sSQLState" has the data type ST_DBSQLError [▶ 89]. The "sSQLState" can supply an errorcode for each database type.

At the following link you can find a list of SQLStates und their discribtion: http://msdn.microsoft.com/en-us/library/ms714687(VS.85).aspx (SQLStates)

Database Type	Errorcodereference
Microsoft SQL Compact Database	http://technet.microsoft.com/en-us/library/ms171788.aspx / TcDBServer_OleDB_Errorcodes.htm [▶ 136]
Microsoft SQL Database	TcDBServer_OleDB_Errorcodes.htm [▶ 136]
Microsoft Access Database	TcDBServer_OleDB_Errorcodes.htm [▶ 136]
MySQL Database	dev.mysql.com/doc/refman/5.0/en/error-handling.html
Oracle Database	http://www.ora-code.com
DB2 Database	https://www.ibm.com/docs/en/db2-for-zos/12?topic=codes-sql
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 [▶ 140]
ASCII Database	TcDBServer_ASCII_Errorcodes.htm [▶ 140]

9.1.3 OleDB Errorcodes

HRESULT	Describtion
0x80040E00	Invalid accessor
0x80040E01	Creating another row would have exceeded the total number of active rows supported by the rowset
0x80040E02	Unable to write with a read-only accessor
0x80040E03	Given values violate the database schema
0x80040E04	Invalid row handle
0x80040E05	An object was open
0x80040E06	Invalid chapter
0x80040E07	A literal value in the command could not be converted to the correct type due to a reason other than data overflow
0x80040E08	Invalid binding info

HRESULT	Describtion
0x80040E09	Permission denied
0x80040E0A	Specified column does not contain bookmarks or chapters
0x80040E0B	Some cost limits were rejected
0x80040E0C	No command has been set for the command object
0x80040E0D	Unable to find a query plan within the given cost limit
0x80040E0E	Invalid bookmark
0x80040E0F	Invalid lock mode
0x80040E10	No value given for one or more required parameters
0x80040E11	Invalid column ID
0x80040E12	Invalid ratio
0x80040E13	Invalid value
0x80040E14	The command contained one or more errors
0x80040E15	The executing command cannot be canceled
0x80040E16	The provider does not support the specified dialect
0x80040E17	A data source with the specified name already exists
0x80040E18	The rowset was builtover a live data feed and cannot be restarted
0x80040E19	No key matching the described characteristics could be found within the current range
0x80040E1A	Ownership of this tree has been given to the provider
0x80040E1B	The provider is unable to determine identity for newly inserted rows
0x80040E1C	No nonzero weights specified for any goals supported, so goal was rejected; current goal was not changed
0x80040E1D	Requested conversion is not supported
0x80040E1E	IRowsOffset would position you past either end of the rowset, regardless of the cRows value specified; cRowsObtained is 0
0x80040E1F	Information was requested for a query, and the query was not set
0x80040E20	Provider called a method from IRowsetNotify in the consumer and NT
0x80040E21	Errors occurred
0x80040E22	A non-NULL controlling IUnknown was specified and the object being created does not support aggregation
0x80040E23	A given HROW referred to a hard- or soft-deleted row
0x80040E24	The rowset does not support fetching backwards
0x80040E25	All HROWs must be released before new ones can be obtained
0x80040E26	One of the specified storage flags was not supported
0x80040E27	The comparison operator was invalid
0x80040E28	The specified status flag was neither DBCOLUMNSTATUS_OK nor DBCOLUMNSTATUS_ISNUL
0x80040E29	The rowset cannot scroll backwards
0x80040E2A	Invalid region handle
0x80040E2B	The specified set of rows was not contiguous to or overlapping the rows in the specified watch region
0x80040E2C	A transition from ALL* to MOVE* or EXTEND* was specified
0x80040E2D	The specified region is not a proper subregion of the region identified by the given watch region handle
0x80040E2E	The provider does not support multi-statement commands
0x80040E2F	A specified value violated the integrity constraints for a column or table
0x80040E30	The given type name was unrecognized
0x80040E31	Execution aborted because a resource limit has been reached; no results have been returned
0x80040E32	Cannot clone a command object whose command tree contains a rowset or rowsets
0x80040E33	Cannot represent the current tree as text

HRESULT	Describtion
0x80040E34	The specified index already exists
0x80040E35	The specified index does not exist
0x80040E36	The specified index was in use
0x80040E37	The specified table does not exist
0x80040E38	The rowset was using optimistic concurrency and the value of a column has been changed since it was last read
0x80040E39	Errors were detected during the copy
0x80040E3A	A specified precision was invalid
0x80040E3B	A specified scale was invalid
0x80040E3C	Invalid table ID
0x80040E3D	A specified type was invalid
0x80040E3E	A column ID was occurred more than once in the specification
0x80040E3F	The specified table already exists
0x80040E40	The specified table was in use
0x80040E41	The specified locale ID was not supported
0x80040E42	The specified record number is invalid
0x80040E43	Although the bookmark was validly formed, no row could be found to match it
0x80040E44	The value of a property was invalid
0x80040E45	The rowset was not chaptered
0x80040E46	Invalid accessor
0x80040E47	Invalid storage flags
0x80040E48	By-ref accessors 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 given accessor was write-only
0x80040E4D	Authentication failed
0x80040E4E	The change was canceled during notification; no columns are changed
0x80040E4F	The rowset was single-chaptered and the chapter was not released
0x80040E50	Invalid source handle
0x80040E51	The provider cannot derive parameter info and SetParameterInfo has not been called
0x80040E52	The data source object is already initialized
0x80040E53	The provider does not support this method
0x80040E54	The number of rows with pending changes has exceeded the set limit
0x80040E55	The specified column did not exist
0x80040E56	There are pending changes on a row with a reference count of zero
0x80040E57	A literal value in the command overflowed the range of the type of the associated column
0x80040E58	The supplied HRESULT was invalid
0x80040E59	The supplied LookupID was invalid
0x80040E5A	The supplied DynamicErrorID was invalid
0x80040E5B	Unable to get visible data for a newly-inserted row that has not yet been updated
0x80040E5C	Invalid conversion flag
0x80040E5D	The given parameter name was unrecognized
0x80040E5E	Multiple storage objects can not be open simultaneously
0x80040E5F	The requested filter could not be opened
0x80040E60	The requested order could not be opened
0x80040E61	Bad tuple
0x80040E62	Bad coordinate

HRESULT	Describtion
0x80040E63	The given axis was not valid for this Dataset
0x80040E64	One or more of the given cell ordinals was invalid
0x80040E65	The supplied columnID was invalid
0x80040E67	The supplied command does not have a DBID (Note: DBID is SQL shorthand for Database ID.)
0x80040E68	The supplied DBID already exists
0x80040E69	The maximum number of Sessions supported by the provider has already been created. The consumer must release one or more currently held Sessions before obtaining a new Session object
0x80040E72	The index ID is invalid
0x80040E73	The initialization string does not conform to specification
0x80040E74	The OLE DB root enumerator did not return any providers that matched an of the SOURCES_TYPEs requested
0x80040E75	The initialization string specifies a provider which does not match the currently active provider.
0x80040E76	The specified DBID is invalid
0x80040E6A	Invalid trustee value
0x80040E6B	The trustee is not for the current data source
0x80040E6C	The trustee does not support memberships/collections
0x80040E6D	The object is invalid or unknown to the provider
0x80040E6E	No owner exists for the object
0x80040E6F	The access entry list supplied is invalid
0x80040E70	The trustee supplied as owner is invalid or unknown to the provider
0x80040E71	The permission supplied in the access entry list is invalid
0x80040E77	The ConstraintType was invalid or not supported by the provider.
0x80040E78	The ConstraintType was not CONSTRAINTTYPE_FOREIGNKEY and cForeignKeyColumns was not zero
0x80040E79	The Deferrability was invalid or the value was not supported by the provider
0x80040E80	The MatchType was invalid or the value was not supported by the provider
0x80040E8A	The UpdateRule or DeleteRule was invalid or the value was not supported by the provider
0x80040E8B	The pConstraintID did not exist in the data source
0x80040E8C	The dwFlags was invalid
0x80040E8D	The rguidColumnType pointed to a GUID that does not match the object type of this column or this column was not set
0x80040E8E	The requested URL was out-of-scope
0x80040E90	The provider could not drop the object
0x80040E91	There is no source row
0x80040E92	The OLE DB object represented by this URL is locked by one or more other processes
0x80040E93	The client requested an object type that is only valid for a collection
0x80040E94	The caller requested write access to a read-only object
0x80040E95	The provider could not connect to the server for this object
0x80040E96	The provider could not connect to the server for this object
0x80040E97	The attempt to bind to the object timed out
0x80040E98	The provider was unable to create an object at this URL because an object named by this URL already exists
0x80040E99	The provider could not drop the object
0x80040E9A	The provider was unable to create an object at this URL because the server was out of physical storage
0x00040EC0	Fetching requested number of rows would have exceeded total number of active rows supported by the rowset

HRESULT	Describtion
0x00040EC1	One or more column types are incompatible; conversion errors will occur during copying
0x00040EC2	Parameter type information has been overridden by caller
0x00040EC3	Skipped bookmark for deleted or non-member row
0x00040EC4	Errors found in validating tree
0x00040EC5	There are no more rowsets
0x00040EC6	Reached start or end of rowset or chapter
0x00040EC7	The provider re-executed the command
0x00040EC8	Variable data buffer ful
0x00040EC9	There are no more results
0x00040ECA	Server cannot release or downgrade a lock until the end of the transaction
0x00040ECB	Specified weight was not supported or exceeded the supported limit and was set to 0 or the supported limit
0x00040ECC	Consumer is uninterested in receiving further notification calls for this reason
0x00040ECD	Input dialect was ignored and text was returned in different dialect
0x00040ECE	Consumer is uninterested in receiving further notification calls for this phase
0x00040ECF	Consumer is uninterested in receiving further notification calls for this reason
0x00040ED0	The operation is being processed asynchronously
0x00040ED1	In order to reposition to the start of the rowset, the provider had to reexecute the query; either the order of the columns changed or columns were added to or removed from the rowset
0x00040ED2	The method had some errors; errors have been returned in the error array
0x00040ED3	Invalid row handle
0x00040ED4	A given HROW referred to a hard-deleted row
0x00040ED5	The provider was unable to keep track of all the changes; the client must refetch the data associated with the watch region using another method
0x00040ED6	Execution stopped because a resource limit has been reached; results obtained so far have been returned but execution cannot be resumed
0x00040ED7	The bind failed because the provider was unable to satisfy all of the bind flags or properties
0x00040ED8	A lock was upgraded from the value specified
0x00040ED9	One or more properties were changed as allowed by provider
0x00040EDA	Errors occurred
0x00040EDB	A specified parameter was invalid
0x00040EDC	Updating this row caused more than one row to be updated in the data source
0x00040EDD	The row has no row-specific columns

9.1.4 ASCII Errorcodes

Code	Description
1	Function not allowed.
2	Syntax error
3	File couldn't be open.

9.1.5 XML Errorcodes

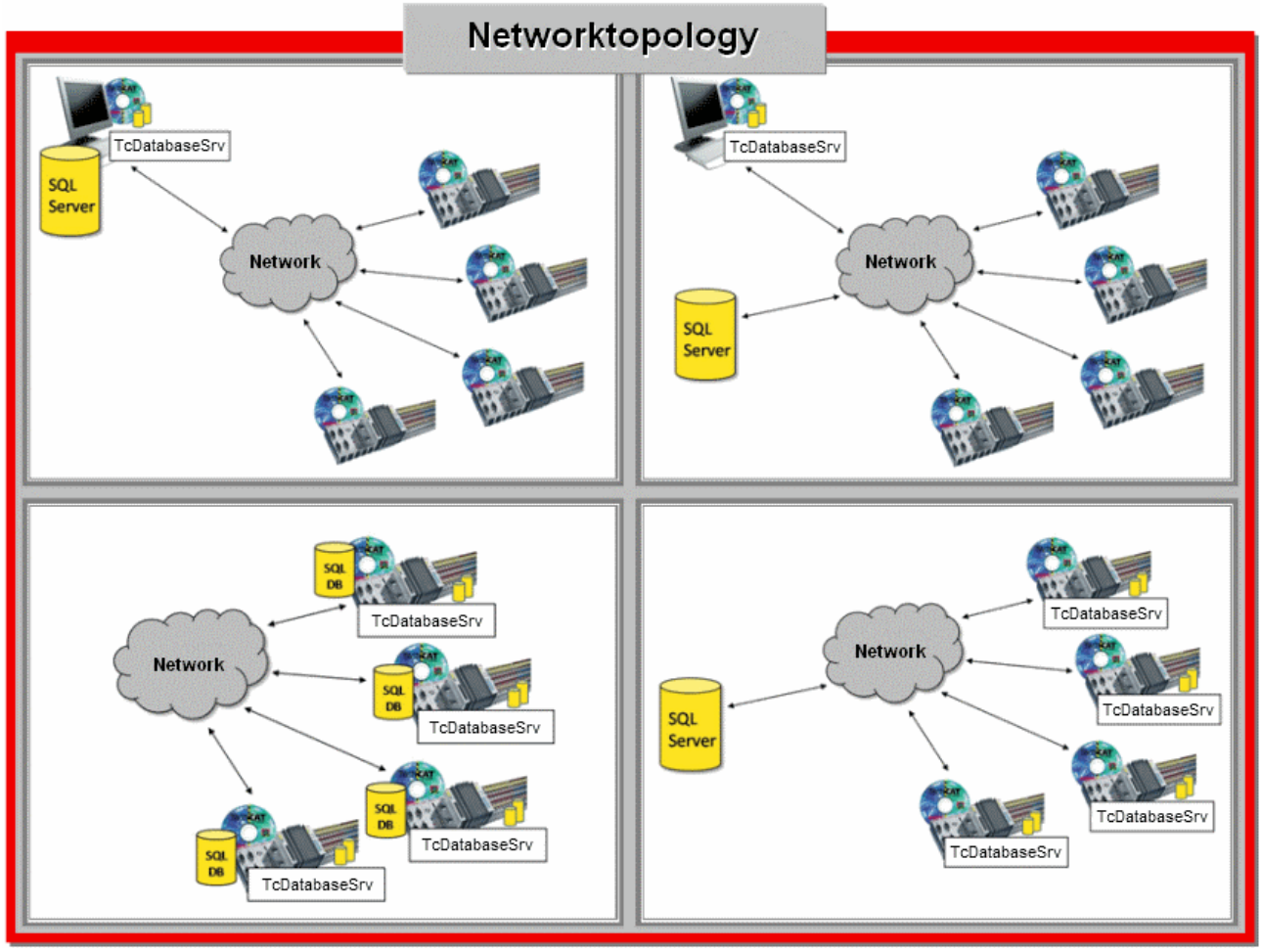
Code	Description
1	Function not allowed.
2	XML file couldn't be read
3	XML schema couldn't be read
4	Syntax error

Code	Description
5	Table couldn't be created.
6	The list of INSERT VALUES are not conform to the column list
7	Memory of the PLC structure not big enough.
8	XML file couldn't be created.
9	XML database not found.
10	XML table not found.

9.2 Network topology

The TwinCAT Database Server can be carried out in the network in several ways. The network topology mostly will be controlled by the database type, the local terms, and the area of application.

In the following figure you can see several network topologies where the TwinCAT Database Server can be applied.



Important!

For a smooth remote access of the TwinCAT Database Servers to a database some things must be regarded at the database side:

- Is the remote access allowed in general?
- Note the quorum of the concurrent opened connections. (If the TwinCAT Database Server needs to open several connections)
- Has the user who wants to register enough rights
- Is the Firewall of the Remote Systems configurated?

For furthermore details how to configure your database server you can find in the corresponding documentation of the database.

9.3 FAQ - Frequently asked questions and their answers

In this area we answer frequently asked questions to help you to work with TwinCAT Database Server.
If you have any further questions, please contact our support (-157)

1. Which performance can be achieved by the TwinCAT Database Server? [▶ 142]
2. Supports the TwinCAT Database Server Stored Procedures? [▶ 142]
3. Which data types are supported by the TwinCAT Database Server? [▶ 142]
4. Is it possible to log more than one variable of a symbol group in one data record? [▶ 142]
5. How do I write or read single variables out of an existing database structure? [▶ 142]
6. Is it possible to log several data records at the same time into a database? [▶ 142]
7. Which network topologies are supported by the TwinCAT Database Server? [▶ 142]
8. Which functionalities of the TwinCAT Database Server can be used for the database type "XML"? [▶ 142]
9. Why are some Function Blocks in an "Obsolete" folder? [▶ 143]
10. Which databases, supported by the TwinCAT Database Server, could be connected over network? [▶ 143]

?Which performance can be achieved by the TwinCAT Database Server?

! That's much too sweeping a statement. The performance depends on the used hardware, the used write mode e.g., ring buffer mode and the count of the variable which must be logged. Another aspect is the used database type.

?Supports the TwinCAT Database Server Stored Procedures?
! Yes, the TwinCAT Database Server support Stored Procedures with the help of the PLC function blocks FB_DBStoredProcedures [▶ 78] and FB_DBStoredProceduresRecordReturn [▶ 86]. These function blocks are not supporte by every database type.

?Which data types are supported by the TwinCAT Database Server?
! This Link [▶ 35] shows all supported databases of the TwinCAT Database Server.

?Is it possible to log more than one variable of a symbol group in one data record?
! Symbol groups will be created at the TwinCAT Database Server Configuration Editor. The declared symbols can only be logged separately into the database. To log several variable into one data record, use the function block FB_DBRecordInsert_Ex [▶ 75] out of the PLC.

?How do I write or read single variable out of an existing database structure?
! With the help of the PLC function block FB_DBRecordInsert_Ex [▶ 75] it is possible to write single variable into an existing database structure. Reading of single variable can be done by the function block FB_DBRecordSelect_Ex [▶ 85].

?Is it possible to log several data records at the same time into a database?
! This depends on the used database. The database type "Microsoft SQL Database" supports this in conjunction with the function block FB_DBRecordInsert_Ex. You only must seperate the different SQL INSERT commands with ";".

?Which network topologies are supported by the TwinCAT Database Server?
! It exists several possibilities to use the TwinCAT3 Database Server in a network. This Link [▶ 141] shows the different supported network topologies und further information about this topic.

?Which functionalities of the TwinCAT Database Server can be used for the database type "XML"?
! The database type "XML" supports the full functionality of the TwinCAT Database Server. Only Stored Procedures and Delete SQL commands are not supported. You can work with the XML file like every other database with SQL commands, or with the cyclic write mode of the PLC values. An additional functionality is the possibility to use XPath commands and read XML-Tags. For further information look here [▶ 48].

?Why are some Function Blocks in an "Obsolete" folder?
! During the product development it gives new PLC function blocks which include the functionallity of older function blocks. Specially in new projects it makes no sense to use the obsolete FB's. Of course, the old function blocks are still part of the product. Here are the details:

- *FB_DBAuthentificationAdd* could be replaced by **FB_DBConnectionAdd**
- *FB_DBRecordInsert* could be replaced by **FB_DBRecordInsert_Ex**
- *FB_DBRecordSelect* und *FB_DBRecordSelect_Ex* could be replaced by **FB_DBRecordArraySelect**
- *FB_DBStoredProcedureRecordReturn* could be replaced by **FB_DBStoredProcedureRecordReturn**

?Which databases, supported by the TwinCAT Database Server, could be connected over network?
! It is not possible to reach all databases, which are supported by the TwinCAT Database Server, over network. Here a list of all databases which must be used locally on device:

- Microsoft Access Database
- Microsoft SQL Compact Database
- XML Database
- ASCII File

All other supported databases can be connected over network!

More Information:
www.beckhoff.com/ts6420

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