# Table of contents

1 Foreword ............................................................................................................................... 5  
1.1 Notes on the documentation ............................................................................................. 5  
1.2 Safety instructions .............................................................................................................. 6  

2 Overview .................................................................................................................................. 7  

3 Installation .................................................................................................................................. 8  
3.1 System requirements .......................................................................................................... 8  
3.2 Installation ........................................................................................................................... 8  
3.3 Licensing ............................................................................................................................. 11  
3.4 Installation Windows CE .................................................................................................... 13  

4 Technical introduction ............................................................................................................ 16  
4.1 Basic concept ...................................................................................................................... 16  
4.2 Areas of application and network topologies .................................................................... 17  
4.3 Compatibility ....................................................................................................................... 19  

5 Configuration ........................................................................................................................ 22  
5.1 Configurator ....................................................................................................................... 22  
5.1.1 Integration in Visual Studio ............................................................................................ 22  
5.1.2 Standalone Configurator ............................................................................................... 99  
5.2 Databases .......................................................................................................................... 122  
5.2.1 General Information ...................................................................................................... 122  
5.2.2 MS SQL database ......................................................................................................... 124  
5.2.3 MS SQL Compact database ........................................................................................... 126  
5.2.4 MySQL database .......................................................................................................... 127  
5.2.5 Oracle database ............................................................................................................ 129  
5.2.6 SQLite ............................................................................................................................ 130  
5.2.7 ASCII-File ...................................................................................................................... 131  
5.2.8 XML database .............................................................................................................. 132  
5.2.9 ODBC databases .......................................................................................................... 138  
5.2.10 MS Access database ................................................................................................... 145  
5.2.11 MS Excel database ...................................................................................................... 146  
5.2.12 MongoDB ................................................................................................................... 147  
5.2.13 PostgreSQL ................................................................................................................ 149  
5.2.14 InfluxDB ....................................................................................................................... 150  

6 PLC API .................................................................................................................................... 152  
6.1 Tc3_Database .................................................................................................................... 152  
6.1.1 Function blocks .............................................................................................................. 152  
6.1.2 Data types ..................................................................................................................... 222  
6.1.3 Global constants ........................................................................................................... 243  
6.1.4 Obsolete ......................................................................................................................... 243  
6.2 Tc2_Database .................................................................................................................... 290  
6.2.1 Function blocks .............................................................................................................. 292  
6.2.2 Data types ..................................................................................................................... 319  
6.2.3 Global Constants .......................................................................................................... 323  

7 Examples .................................................................................................................................. 325
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Scenario examples</td>
<td>325</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Best practices</td>
<td>345</td>
</tr>
<tr>
<td>7.2</td>
<td>Tc2_Database</td>
<td>357</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Creating an MS Access database</td>
<td>358</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Starting / stopping, cyclic logging</td>
<td>361</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Logging of a PLC variable with FB_DBWrite</td>
<td>362</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Example with the FB_DBRecordInsert and FB_DBRecordSelect function blocks</td>
<td>366</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Stored procedures with FB_DBStoredProceduresRecordArray</td>
<td>368</td>
</tr>
<tr>
<td>7.2.6</td>
<td>Using XML as database</td>
<td>371</td>
</tr>
<tr>
<td>7.2.7</td>
<td>XPath sample to illustrate the different SELECT types</td>
<td>376</td>
</tr>
<tr>
<td>7.2.8</td>
<td>XPath sample with XML schema</td>
<td>379</td>
</tr>
<tr>
<td>8</td>
<td>Appendix</td>
<td>384</td>
</tr>
<tr>
<td>8.1</td>
<td>Error codes</td>
<td>384</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Tc3_Database</td>
<td>384</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Tc2_Database</td>
<td>394</td>
</tr>
<tr>
<td>8.2</td>
<td>FAQ - frequently asked questions and answers</td>
<td>405</td>
</tr>
<tr>
<td>8.3</td>
<td>Support and Service</td>
<td>406</td>
</tr>
</tbody>
</table>
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:
with corresponding applications or registrations in various other countries.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Description of symbols

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

<table>
<thead>
<tr>
<th><strong>DANGER</strong></th>
<th>Serious risk of injury!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>WARNING</strong></th>
<th>Risk of injury!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CAUTION</strong></th>
<th>Personal injuries!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>NOTE</strong></th>
<th>Damage to the environment or devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
<td></td>
</tr>
</tbody>
</table>

Tip or pointer

This symbol indicates information that contributes to better understanding.
2 Overview

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

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

Components

- **TwinCAT Database Server [16]**: The service is started and stopped together with TwinCAT. It forms the link between the TwinCAT system and the database.
- **Configurator [99]**: The TwinCAT Database Server Configurator facilitates visual setting of the database parameters required for basic communication with the respective database.
- **PLC library [290]**: The PLC library includes various function blocks. They enable establishment of a database connection, creation of a new table, writing of data into any table structure using Insert commands, and reading via Select commands. It is also possible to update or delete database entries and trigger stored procedures. NoSQL databases have their own function blocks that are optimized for handling flexible JSON documents in the PLC, for example. The principle of operation is identical.

Principle of operation

Within the TwinCAT system the Database Server communicates via ADS. Externally it links to the respective configured database. Possible network topologies can be found in section "Areas of application and network technologies [17]".
3 Installation

3.1 System requirements

<table>
<thead>
<tr>
<th>Technical data</th>
<th>TF6420 TwinCAT 3 database server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target system</td>
<td>Windows 7, Windows 8, Windows 10, WinCE PC (x86, x64 and ARM)</td>
</tr>
<tr>
<td>.NET Framework</td>
<td>.Net 4.5.1 or higher</td>
</tr>
<tr>
<td></td>
<td>WinCE: .NET 3.5</td>
</tr>
<tr>
<td>Min. TwinCAT version</td>
<td>3.1.4018</td>
</tr>
<tr>
<td>Min. TwinCAT level</td>
<td>TC1200 TC3</td>
</tr>
</tbody>
</table>

3.2 Installation

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

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

1. Run the setup file as administrator. To do this, select the command Run as administrator in the context menu of the file.
   ➔ The installation dialog opens.

2. Accept the end user licensing agreement and click Next.
3. Enter your user data.

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

![Screenshot of installation dialog box](image1.png)

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

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

![Screenshot of TwinCAT installation confirmation dialog box](image2.png)
7. Select **Finish** to exit the setup.

![Beckhoff Setup Completed](image)

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

### 3.3 Licensing

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

#### Licensing the full version of a TwinCAT 3 Function

A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

#### Licensing the 7-day test version of a TwinCAT 3 Function

- **i** A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

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

 ⇒ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.
4. In the Solution Explorer, double-click License in the SYSTEM subtree.

4.\(^\text{a}\) The TwinCAT 3 license manager opens.

5. Open the Manage Licenses tab. In the Add License column, check the check box for the license you want to add to your project (e.g. “TF4100 TC3 Controller Toolbox”).

5.\(^\text{a}\) In the tabular overview of licenses, the previously selected license is displayed with the status “missing”.

6. Open the Order Information (Runtime) tab.

6.\(^\text{a}\) In the tabular overview of licenses, the previously selected license is displayed with the status “missing”.

\(^\text{a}\) Licenses are ordered alphabetically.
7. Click **7-Day Trial License...** to activate the 7-day trial license.

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

8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.

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

10. Restart the TwinCAT system.

The 7-day trial version is enabled.

### 3.4 Installation Windows CE

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

1. Download and install the setup file [13]
2. Transfer the CAB file to the Windows CE device [14]
3. Run the CAB file on the Windows CE device [14]

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

- Software upgrade [14]

**Download and install the setup file**

The CAB installation file for Windows CE is part of the TFxxx setup. This is made available on the Beckhoff website www.beckhoff.com and automatically contains all versions for Windows XP, Windows 7 and Windows CE (x86 and ARM).
Download the TFxxx setup file and install the TwinCAT 3 function as described in the Installation section.

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

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

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

Example: "TF6310" installation folder

Transfer the CAB file to the Windows CE device

Transfer the corresponding CAB file to the Windows CE device.

There are various options for transferring the executable file:

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

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

Run the CAB file on the Windows CE device

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

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

The software is installed in the following directory on the Windows CE device:

```
\Hard Disk\TwinCAT\Functions\TFxxxx
```
4. Restart the Windows CE device.
5. Transfer the new CAB file to the Windows CE device.
6. Run the CAB file on the Windows CE device and install the new version.
7. Delete the file \texttt{Tc*.old}.
8. Restart the Windows CE device.
\(\Rightarrow\) The new version is active after the restart.
4 Technical introduction

4.1 Basic concept

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

- **Configure mode: pure configuration solution**
  Database connections for simple applications based on graphical configurations without code implementation.

- **PLC Expert mode: programming solution for conventional PLC programmers**
  Database connection for simple or complex applications based on PLC function blocks, in which the database commands are largely generated automatically by the Database Server.

- **SQL Expert mode: programming solution for conventional PLC programmers and database experts**
  Database connection for simple or complex applications based on PLC function blocks and C++ interfaces, in which the database commands are assembled automatically during program execution. For maximum flexibility.

- **NoSQL Expert mode: programming solution for PLC programmers and NoSQL database experts**
  Database connection for simple to complex applications with PLC function blocks, in which NoSQL commands can be created and sent within the program sequence.

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

The TwinCAT Database Server can be set up via a graphical configurator. The configuration is written to an XML file, which can then be downloaded to the target system.

On non-Windows CE devices the configuration file is in folder `C:\TwinCAT\3.1\Boot`, on Windows CE devices in folder `\Hard Disk\TwinCAT\3.1\Boot`.

Read and write access are available for the different database systems. Deselectable database extensions are available for this purpose. The supported databases are described in section "Databases [122]."

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

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

PLC Expert mode

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

SQL Expert Mode

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

• Logging of structures

Note the corresponding byte alignment when logging structures.

NoSql Expert Mode

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

4.2 Areas of application and network topologies

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

The network topology is mostly influenced by the database type, the local conditions and the area of application. The following illustration shows various network topologies in which the TwinCAT Database Server can be used.
1. TwinCAT and the TwinCAT Database Server reside on the same computer, together with the database. The Database Server can act as gateway for many controllers via ADS. The performance must be taken into account.

2. TwinCAT and the TwinCAT Database Server reside on the same computer, while the database is on an external device. Here, too, the Database Server can act as gateway for many controllers via ADS. The performance must be taken into account.

3. The TwinCAT Database Server resides locally on each control device that has a database installed. Not all databases are suitable for this kind of application.

4. This is the most common use case. The TwinCAT Database Server is installed on each control device, and the database resides on an external server in the network.

5. Combination of case 3 and case 4. A main database resides on a server in the network, and the controllers in the field each have a local database, which kicks in when a disconnection is encountered, for example, and stores the data locally in the first instance. The Database Server is installed on each control device.
Remote access by the TwinCAT Database Server to a database

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

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

More detailed information about the configuration of your database server can be found in the corresponding database documentation [122].

4.3 Compatibility

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

The TwinCAT database connections have previously been available in versions 3.0.x, 3.1.x and 3.2.x. The current version is 3.3.x. As before, the database server consists of the following components: configurator, ADS server and PLC library. Version 3.0.x includes the PLC library Tc2_Database.compiled library. The PLC library in versions 3.1.x and higher is called Tc3_Database.compiled-library.

Overview of released Database Server versions

<table>
<thead>
<tr>
<th>Database Server 3.0.x</th>
<th>3.0.23</th>
<th>3.0.26</th>
<th>3.0.27</th>
<th>3.0.28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Server 3.1.x</td>
<td></td>
<td></td>
<td>3.1.29</td>
<td>3.1.30</td>
</tr>
<tr>
<td>Database Server 3.2.x</td>
<td></td>
<td></td>
<td></td>
<td>3.2.32</td>
</tr>
<tr>
<td>Database Server 3.3.x</td>
<td></td>
<td></td>
<td></td>
<td>3.3.33</td>
</tr>
</tbody>
</table>

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

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

Backup of the old XML configuration

During an update from the TwinCAT Database Server 3.0.x to the new 3.1.x version, the old XML configuration is saved. It is renamed to "CurrentConfigDataBase_OLD.xml" and remains in the TwinCAT boot directory.
Notwithstanding the general compatibility referred to above, an old configurator and an old ADS server (version 3.0.x) cannot be used with the new Tc3_Database.compiled library. The diagram below provides a compatibility overview.

<table>
<thead>
<tr>
<th>Konfigurator 3.1.x</th>
<th>Server 3.0</th>
<th>Tc2_Database.compiled-library</th>
<th>Konfigurator 3.1.x</th>
<th>Server 3.1</th>
<th>Tc3_Database.compiled-library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Visual Studio</td>
<td>✔</td>
<td>❌</td>
<td>✔</td>
<td>❌</td>
<td>✔</td>
</tr>
<tr>
<td>Server 3.1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Tc3_Database.compiled-library</td>
<td>❌</td>
<td>❌</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

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

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

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

**Example:**

Note for version 3.1.x: FB_SQLCommand
Note for version 3.2.x: FB_SQLCommandEvt

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

Notes on the transition from 3.2.x to 3.3.x

With version 3.3.x, the modularization of the TwinCAT Database Server has been further advanced on the driver side. Full compatibility is ensured. However, some drivers are no longer automatically included in the setup and must be made available separately by the user. Example: MySQL under Windows CE. For details, please refer to the settings for the respective database in the Configuration section.
5 Configuration

5.1 Configurator

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

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

5.1.1 Integration in Visual Studio

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

5.1.1.1 General

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

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

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

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

- Red: The TwinCAT Database Server cannot be reached.
- Blue: The TwinCAT Database Server has no valid license. (See Licensing [11])
- Green: The TwinCAT Database Server can be reached and is ready for use.
- Violet: The TwinCAT Database Server is in AutoLog mode. Data are exchanged between PLC and database.
The TwinCAT Database projects map a file-based project system. The individual configuration documents are managed in the Solution Explorer. Any modifications that are pending in the editors are identified with * in the documents. If changes are made without opening a document (through the Properties window), the changes are nevertheless registered. Further information on the project system can be found in section "TwinCAT Database Server project [26]."

**Toolbar and commands**

The toolbar has the following elements:
<table>
<thead>
<tr>
<th>Toolstrip button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Activation of the configuration</td>
</tr>
<tr>
<td></td>
<td>Read configuration of the target device</td>
</tr>
<tr>
<td></td>
<td>Save configuration in an XML file</td>
</tr>
<tr>
<td></td>
<td>Read configuration from an XML file</td>
</tr>
<tr>
<td></td>
<td>Add new database configuration</td>
</tr>
<tr>
<td></td>
<td>Add new AutoLog group</td>
</tr>
<tr>
<td></td>
<td>Event display</td>
</tr>
<tr>
<td></td>
<td>Database pool</td>
</tr>
<tr>
<td></td>
<td>AutoLog Viewer</td>
</tr>
<tr>
<td></td>
<td>InformationLog View</td>
</tr>
<tr>
<td></td>
<td>SQL Query Editor</td>
</tr>
</tbody>
</table>

**Properties window**

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

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

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

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

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

5.1.1.1.2 TwinCAT Database Server Project

Build Project

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

To create a new TwinCAT Connectivity project, select Empty TwinCAT Connectivity Project, specify the project name and the storage location and click OK to add it to the solution. In this way, TwinCAT Connectivity projects or TwinCAT Database Server projects can conveniently be created in parallel with TwinCAT or other Visual Studio projects.
A new project node appears in the solution. Below the Connectivity project node you can add subprojects for the supported connectivity functions.

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

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

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

**Editor for server settings**

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

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the **Information Log Viewer**. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.
For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

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

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

**Supported database types**

The installed database types can be selected in the server settings. All installed databases are selected by default. The TwinCAT 3 Database Server will load the corresponding database interfaces. In this way, unused databases on the target system can be deselected.

**Server settings in the Properties window**

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

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

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

A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.
The new database configuration in the TwinCAT Database Server project can be edited via the Properties window or a special editor:

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

Editor for database configurations

```
Connection String:
Server=172.17.64.146; Port=3306; Database=testdb_mysql; Uid=TestUser; Pwd=XXXXXXXXXX;
```
The database ID, which is required for some function blocks in the PLC, is shown in the upper part of the editor (1). The database types of the target database can be selected from the drop-down menu (2). Another option is the ODBC interface for a database, although this is not yet supported. Note that not all functions of the TwinCAT Database Server can be guaranteed, depending on the database.

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

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

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

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

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

**Copying a database configuration into the database pool**

A corresponding command is available in the context menu for copying a database configuration into the database pool (51). It is also possible to use drag & drop to move database configurations between the project and the database pool.
### Deactivating database configurations

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

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

A new AutoLog group and the corresponding components are added as files to the project folder and integrated in the project. They include the ADS device, the symbol groups and the table settings. In order to save these files in the project, you should save the TwinCAT Connectivity project file. The files can then be edited in editors or in the Properties window.
AutoLog mode can be enabled manually (with a command in the PLC or from the configurator) or automatically during system startup.

**Direction**
The set ADS device is used as data target or data source.

**Write mode**
The data can appended in a database line-by-line, held in a ring buffer on a temporal or quantitative basis, or simply be updated at the corresponding position.

**Ring buffer parameter**
Depending on the setting this parameter represent the time or the cycles after which the ring buffer is updated.

**Log mode**
The variable is written either after a certain cycle time or when a change occurs.

**Cycle Time**
Cycle time after which the variable is written.

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

**AdsDevice**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS Device</td>
<td>local</td>
</tr>
<tr>
<td>AMS NetID</td>
<td>5.19.111.106.1.1</td>
</tr>
<tr>
<td>AMS Port</td>
<td>851</td>
</tr>
<tr>
<td>Timeout</td>
<td>2000</td>
</tr>
<tr>
<td>Connection Type</td>
<td>bySymbolName</td>
</tr>
</tbody>
</table>

**Configuring symbols**

The symbols you set here are written to or read from the database, depending on whether the ADS device is the data target or the data source. The TwinCAT Target browser [52] can be used for convenient access. Here you can search for the symbols on the target and communicate between the two tools via drag & drop.

<table>
<thead>
<tr>
<th>ADS Device</th>
<th>Name of the ADS target device.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMS NetID</td>
<td>Address of the target device in the TwinCAT network.</td>
</tr>
<tr>
<td>AMS Port</td>
<td>Port of the target device in the TwinCAT network.</td>
</tr>
<tr>
<td>Timeout</td>
<td>Time after which it is assumed that the connection to the target device is lost.</td>
</tr>
<tr>
<td>Connection Type</td>
<td>bySymbolName: Connection is established based on the symbol name. byIndexGroup: Connection is established based on the memory index.</td>
</tr>
</tbody>
</table>
Symbols can also be added manually to symbol groups or edited. The information that is required varies, depending on whether in the ADS device the connection type was selected via the symbol name or the index groups. The starting point is always the ADS device.
SymbolName | The symbol is addressed based on the set ADS device  
Symbol database name | Name of the variable in the database table  
DataType | PLC data type of the symbol  
BitSize | Bit size of the symbols (set automatically for the data types)  
IndexGroup | Index group in the TwinCAT system  
IndexOffset | Index offset in the TwinCAT system

### Configuring a table

The table in a database can be based on a standard table structure or on an individual structure. The corresponding table can be selected from a list of possible tables. If the table does not yet exist, you can create it via the SQL Query Editor. If you select the standard table structure, a blue tick indicates whether the selected table corresponds to this structure.

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

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

The SQL Query Editor is a Database Server tool that supports the development of your application. The tool can be used to test connections and SQL commands and to check the compatibility between PLC and databases.
First select the target system from the routes of your TwinCAT system (1). The TwinCAT Database Server must be installed on the target system. If a NoSQL database is stored in the configuration, an additional NoSQL tab is visible. You will find the documentation in a subitem below.

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

Three commands are available for generating the individual SQL commands:

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

The differences in the individual SQL commands are explained below.

Select command

Select commands can be created and sent via the SELECT tab. Select commands give the opportunity to read records from the databases. After executing the command, values are returned if they exist in the table. They are listed under "Value" in the table structure display. Use the arrows under the display to navigate through the individual records.
The Insert command gives the opportunity to write records into the table. The values under "Value" can be modified once the table structure has been retrieved. If the command is then generated, the values in the Insert command will automatically be in the right format. These values are written into the table when the command is executed.

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

The Delete command has two functions.

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

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

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

The table name can be determined by executing the command. The table with the configured table structure is created.
The TwinCAT Database Server supports "Stored Procedures", which provide numerous databases for processing more complex queries at the database level or to make a simplified interface available.

If Stored Procedures are available in the database and the table, you can list and select them (1). The input and output parameters can be picked up automatically (2) and transferred to the tables in the display (3)(4). The parameter type, name and data type are displayed there. In addition you can insert values here, in order to execute the Stored Procedures with the input values via "Execute". The result is displayed in the output values (4). If several records are returned, the arrow keys can be used to switch between them. This functionality serves as development aid for the call in the PLC. The results are returned there by calling the corresponding function block [198].
NoSql Query Editor

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

Your available databases are then listed in the list of NoSQL databases (3). SQL databases are not displayed in this list. All existing collections are now listed under a selected database. In the menu bar (2) the list can be updated, and collections can be added or deleted.
Once a collection has been selected, the target (6) of the query to be sent changes. The following functions (8) are provided:

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

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

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

**Delete:** Executes a delete query on the data found with the filter in the text field (7). Any data that is found is deleted from the collection.
**Validate**: If this option is selected, the data queries are not automatically parsed according to their own schema, but an attempt is made to map these data to the structure of the symbol from the PLC, which was specified via these parameters.

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Target system</td>
<td>Choose Target System with installed TwinCAT Database Server</td>
</tr>
<tr>
<td>2</td>
<td>Start</td>
<td>Manual start of the AutoLog mode</td>
</tr>
<tr>
<td>3</td>
<td>Login</td>
<td>Logging into the active AutoLog process</td>
</tr>
<tr>
<td>4</td>
<td>Logout</td>
<td>Logging out of the active AutoLog process</td>
</tr>
<tr>
<td>5</td>
<td>Stop</td>
<td>Manual stop of the AutoLog mode</td>
</tr>
<tr>
<td>6</td>
<td>AutoLog groups</td>
<td>List of configured AutoLog groups on the target system</td>
</tr>
<tr>
<td>7</td>
<td>Symbols</td>
<td>List of configured symbols for the selected AutoLog group</td>
</tr>
</tbody>
</table>
5.1.1.7 Database configuration pool

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

<table>
<thead>
<tr>
<th>Database Pool</th>
<th>Type</th>
<th>Date Modified</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbName</td>
<td>SQL Server</td>
<td>09.01.2017 12:32</td>
<td>this database is the default database to collect all sensor data that are distributed to the grid of the consumer.</td>
</tr>
<tr>
<td>dbName_in</td>
<td>ODBC</td>
<td>09.01.2017 12:33</td>
<td>this database is used to collect and measure all relevant data of the consumers in the home, e.g. the temperature and power of the devices.</td>
</tr>
<tr>
<td>dbName_wind</td>
<td>MS Server</td>
<td>09.01.2017 12:34</td>
<td>the standard database for our windfarm park. This database can be used for every windfarm park to collect data about the windmill.</td>
</tr>
<tr>
<td>dbName_fine</td>
<td>MS Server</td>
<td>09.01.2017 12:35</td>
<td>the database for the windmills to collect data if the internet connection gets lost. We don't lose any data anymore.</td>
</tr>
</tbody>
</table>

5.1.1.8 InformationLog View

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

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

The TwinCAT Target Browser is the central data management interface in the TwinCAT 3 Engineering. It can provide data from various TwinCAT target systems live via ADS or access historic data in databases. Depending on what TwinCAT 3 functions are installed on an engineering system, the ADS standard extension of the TwinCAT Target Browser is extended by further extensions (see Extensions [53]).

Call options

Traditionally, the target browser can be called via the Scope menu in Visual Studio®. Due to its ever increasing use in the TwinCAT system, it can also be opened via the TwinCAT menu under the item Target Browser.
In addition, it is possible that the various tools that use the Target Browser will provide additional calls. For example, some products do this via the context menu on their respective project nodes.

**Architecture**

Within Microsoft Visual Studio® the TwinCAT Target Browser is a tool window that is subdivided into two sections. Target systems are displayed on the left-hand side ("Specific Target Area"). You can switch between the various extensions via the tabs. The details of the respective target system or the selected object respectively are displayed on the right-hand side ("Common Symbol Area").

Many extensions support a "Value Preview" for variables. This means that if you select a variable and press and hold the space bar, a small chart appears. In this way you can determine whether data have arrived or exist in a database. A search bar above the right-hand section enables the filtering/reduction of the visible symbols. The entry must be confirmed with the [Enter] key.

In addition, a breadcrumb navigation bar shows you where you are currently located.

**Extensions**

The following table shows an overview of the current extensions and the product from which they were installed. Further information on the extensions can be found in the associated document sections.

<table>
<thead>
<tr>
<th>Extension name</th>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS ![54]</td>
<td>TwinCAT XAE, TwinCAT Scope, TwinCAT IoT Data Agent, TwinCAT Database Server, TwinCAT Analytics Service Tool</td>
</tr>
<tr>
<td>OPC UA ![56]</td>
<td>TwinCAT Scope, TwinCAT IoT Data Agent</td>
</tr>
<tr>
<td>TcAnalytics</td>
<td>TwinCAT Analytics Workbench and Service Tool, TwinCAT Scope</td>
</tr>
<tr>
<td>TcAnalytics File</td>
<td>TwinCAT Analytics Workbench and Service Tool, TwinCAT Scope</td>
</tr>
</tbody>
</table>
Extension – ADS

The TwinCAT Target Browser ADS extension is used most frequently within the TwinCAT system.

Specific Target Area

All target systems registered to the local TwinCAT 3 Engineering are displayed in a tree structure in the left-hand area of the TwinCAT Target Browser (ADS). In first place is the local system, followed by the target systems such as Industrial PCs or Embedded PCs in the order of registration. The prefixed screen symbol indicates the state of the system (green: run mode, blue: config mode, red: stop mode or unreachable). The available default ADS ports are listed below a target system. If you select a port, the available symbols/variables are displayed in the Common Symbol Area on the right-hand side of the TwinCAT Target Browser.

Toolbar

The toolbar of the ADS extension makes the following functions available:
If an ADS port is not displayed by default, further ports can be added to a selection via this command.

If an ADS route to a target system is missing, further target systems can be added via this button.

The display of the target system states can be manually updated with this button.

The ADS symbols available at the selected port are displayed in the right-hand area of the TwinCAT Target Browser. The addresses and the attributes, for example, are also displayed in addition to the name, the data type, the size and the symbol name. Special attributes, such as those of the units, are interpreted and output in their own columns.
Extension - OPC UA

The TwinCAT Target Browser OPC UA extension offers a standardized way into the TwinCAT 3 Engineering.

Specific Target Area

All OPC UA Servers that have been added using the Add command in the toolbar are displayed in a tree structure in the left-hand area of the TwinCAT Target Browser (OpcUa). The screen symbol in front of the server designation at the first level of the tree structure indicates the connection status. Below the server the created end points are subdivided into "Anonymous" (anonymous) and "Authenticated" (user mode). The encryption method is displayed in brackets for each end point. If you select an end point, the available OPC UA Nodes are displayed on the right-hand side of the TwinCAT Target Browser (see also: Displaying OPC UA Nodes [57]).

Toolbar

The toolbar of the OPC UA extension makes the following functions available:

Add

New connections to existing OPC UA Servers can be established with this command (see also: Adding OPC UA Servers [57]).

Remove

A registered server can be removed with this command.

Refresh

The display in the Target Browser tree can be manually updated with this button.

Common Symbol Area

The available OPC UA Nodes are displayed on the right-hand side of the TwinCAT Target Browser. These reflect the hierarchical structure of the PLC project. The node class and the identifier, for example, are also displayed in addition to the name, the data type, the size and the full object name.
Adding OPC UA Servers

1. Click on Add in the OPC UA toolbar.
   - The dialog OpcUa EndPoint Selection opens.

2. Enter the URL of the server.

3. Select the end point from the drop-down list. With OPC UA you can determine via the corresponding end points whether and which method of encryption is used. It is also possible to add several end points to a server. To do this, execute the Add command again.

4. Select whether the access is anonymous or authenticated. If the access is authenticated, enter a user name and password. Authenticated access may be required if a password-protected user management has been set up for the OPC UA server (e.g. different user accounts with different rights).

5. Confirm the dialog.
   - The OPC UA Server is added to the tree structure in the Target Browser with the selected end points.

Displaying OPC UA Nodes

In order to display the available OPC UA Nodes, select the respective end point in the tree structure on the left-hand side. If you select an end point without certified access, the nodes will be displayed directly. If the selected end point is certified, you must first trust the server certificate in a corresponding dialog.
You can trust the certificate in a single case (until the Visual Studio instance is closed) or add it to the list of trusted certificates via the check box **Save Certificate to Trustlist**.

During the first attempt to connect to an OPC UA Server it is additionally necessary to trust the certificate of the client (Target Browser) on the server side. To do this, copy the respective certificate in the certificate directory of the OPC UA Server from the "rejected" folder to the "trusted" folder.

### 5.1.1.1.10 Support Information Report

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

**Plug-in mechanism**

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

**Creating and submitting a Support Information Report**

1. A Support Information Report is open.
2. Use the **Behaviour** text field to describe the behavior that occurred in as much detail as possible.
3. In the **Attachment** area, you can add files (screenshots etc.) to the report via the **Add Attachment** button, if required. Files can optionally be selected via remote access. To do this, select a target from the **Remote System** dropdown list. Depending on the selected target, it may be possible to browse Windows CE devices.
4. Enter your contact details and select a Beckhoff subsidiary for your country. This information is obligatory for submitting the Support Information Report.
5. You will be offered the option to store your contact details for future Support Information Reports. To do this, tick the **Store personal data** check box.
6. The product-specific plug-ins can be found in the lower section of the Support Information Report. Tick the **Include in report** check box. The information required for the product is added automatically, if it is available. The screenshot shows the current configuration of a TwinCAT Database Server in the form of an XML file as an example.
7. Submitting the Support Information Report:
   - If the device has an email connection, you can submit the Support Information Report directly to the Beckhoff subsidiary for your country via the **Send Report** button.
5.1.1.2 Configure mode

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

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

Configure Mode

In Configure mode, the bulk of the work is done in the configurator. The configuration has to be set up for the required database and for the AutoLog group. The target browser can be used for configuring the AutoLog group, for online access to a target system, and for selecting the variables to be communicated. If the **AutoStart** option is used, the communication with the configured database is established directly when TwinCAT system starts up. If the **Manual** option is selected, the communication has to be enabled via the function block **FB_PLCDBAutoLog** [156] or for AutoLog view.
Build Project

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

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

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

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

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

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

**Editor for server settings**

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

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the **Information Log Viewer** [🔗](51). Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.
For network access to file-based databases such as Access or SQL Compact, the Impersonate option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

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

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

**Supported database types**

The installed database types can be selected in the server settings. All installed databases are selected by default. The TwinCAT 3 Database Server will load the corresponding database interfaces. In this way, unused databases on the target system can be deselected.

**Adding a new database configuration**

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

A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.
A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

**Editor for database configurations**

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

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.
For each database additional adjustable parameters are available. Depending on the database a connection string is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

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

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

**Failover database**

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

**Adding a new AutoLog group**

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

A new AutoLog group for the database configuration can be added via the command **Add New AutologGroup** in the context menu of a database configuration or via the toolbar. These AutoLog groups refer to the parent database.
A new AutoLog group and the corresponding components are added as files to the project folder and integrated in the project. They include the ADS device, the symbol groups and the table settings. In order to save these files in the project, you should save the TwinCAT Connectivity project file. The files can then be edited in editors or in the Properties window.
StartUp | AutoLog mode can be enabled manually (with a command in the PLC or from the configurator) or automatically during system startup.
Direction | The set ADS device is used as data target or data source.
Write mode | The data can appended in a database line-by-line, held in a ring buffer on a temporal or quantitative basis, or simply be updated at the corresponding position.
Ring buffer parameter | Depending on the setting this parameter represent the time or the cycles after which the ring buffer is updated.
Log mode | The variable is written either after a certain cycle time or when a change occurs.
Cycle Time | Cycle time after which the variable is written.

**Configuring the ADS device**

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

![AdsDevice](image)

- **Parameter** | **Value**
  - ADS Device | local
  - ADS NetID | 5.19.111.106 1.1
  - ADS Port | 951
  - Timeout | 2000
  - Connection Type | bySymbolName
### Configuration

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

#### Configuring symbols

The symbols you set here are written to or read from the database, depending on whether the ADS device is the data target or the data source. The TwinCAT Target browser [52] can be used for convenient access. Here you can search for the symbols on the target and communicate between the two tools via drag & drop.

Symbols can also be added manually to symbol groups or edited. The information that is required varies, depending on whether in the ADS device the connection type was selected via the symbol name or the index groups. The starting point is always the ADS device.
SymbolName | The symbol is addressed based on the set ADS device
Symbol database name | Name of the variable in the database table
DataType | PLC data type of the symbol
BitSize | Bit size of the symbols (set automatically for the data types)
IndexGroup | Index group in the TwinCAT system
IndexOffset | Index offset in the TwinCAT system

**Configuring a table**

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

The corresponding table can be selected from a list of possible tables. If the table does not yet exist, you can create it via the SQL Query Editor [41]. If you select the standard table structure, a blue tick indicates whether the selected table corresponds to this structure.
The specific table type offers the option to distribute the individual symbols that were set in the symbol group to the table columns in the database as required. When a record is written to the database in AutoLog mode, the current values of the symbol group at the sampling time are saved in the corresponding table column.
Activating a project

To activate a configured project on the TwinCAT Database Server, use the command **Activate Configuration** in the context menu of the TwinCAT Database Server project.
Logging of the variable starts when the TwinCAT system starts, depending on which startup behavior was specified in the AutoLog group. The mode can be started manually via the following AutoLog Viewer or with the corresponding function block from the PLC.

**AutoLog Viewer**

The AutoLog Viewer of the TwinCAT Database Server is a tool for controlling and monitoring the AutoLog mode. You can log into a target system, similar to the TwinCAT PLC. In logged-in state the AutoLog mode can be started or stopped. Information on the current state of the logging is shown in the lower part of the window. When an AutoLog group is selected, further information is displayed via the logged symbols.
The AutoLog Viewer can be used to start and monitor the configured application. Depending on setting, after login and startup the incrementing cycle counter of the AutoLog group is visible according to the update times. Update errors are also shown here. For more detailed handling we recommend the InformationLog View.

**More detailed error handling with the InformationLog View**

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

The log files can not only be viewed or emptied via direct file access, but also directly via the target. This is particularly advantageous with distributed Database Servers in a network, for quick and easy access to the log file. For this access a route to the target device must exist.
This chapter is a compilation of all the information required for using the PLC Expert mode of the TwinCAT Database Server. In contrast to the Configure mode, in this mode data are not written or read based on cycles or events, but at specific times within the program sequence. This requires knowledge of the SQL language.

**PLC Expert mode**

In PLC Expert mode only the database configuration is set in the configurator. Further functionalities are implemented in the PLC code of the application. With the function block FB_PLCDBCreate [168] it is possible to dispense with the configurator and even configure the database itself from the PLC. Function blocks for reading and writing are available, if required. The function block FB_PLCDBCmd [180] forms the transition between PLC Expert mode and SQL Expert mode. Here, table structures can easily be mapped as PLC structures, and an SQL command with placeholders for the current structure values can be transferred to the TwinCAT Database Server. The TwinCAT Database Server then inserts all values automatically and sends the command to the database.

**Build Project**

The TwinCAT Connectivity extension for Visual Studio provides a new project template. When a new project is created, the TwinCAT Connectivity Project category appears as an option.
To create a new TwinCAT Connectivity project, select **Empty TwinCAT Connectivity Project**, specify the project name and the storage location and click **OK** to add it to the solution. In this way, TwinCAT Connectivity projects or TwinCAT Database Server projects can conveniently be created in parallel with TwinCAT or other Visual Studio projects.

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

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

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

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

Editor for server settings

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

The settings for logging faults or errors can be configured under Log settings. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the Information Log Viewer. Under Log Settings you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.
For network access to file-based databases such as Access or SQL Compact, the Impersonate option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

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

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

**Supported database types**

The installed database types can be selected in the server settings. All installed databases are selected by default. The TwinCAT 3 Database Server will load the corresponding database interfaces. In this way, unused databases on the target system can be deselected.

**Adding a database configuration**

A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.
A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

Editor for database configurations

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

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.
For each database additional adjustable parameters are available. Depending on the database a connection string is created, which describes the connection to the database. The intention is to make the parameters more transparent.

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

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

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

Activating a project

To activate a configured project on the TwinCAT Database Server, use the command Activate Configuration in the context menu of the TwinCAT Database Server project.
Once the project has been activated, the SQL Query Editor can be used for further development steps, such as creating databases or tables, generating structures for the PLC, which match the corresponding table structure of the database, or testing connections to the database with the implemented information.

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

5.1.1.4 SQL Expert mode

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

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

SQL Expert Mode

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

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

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

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

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

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

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

**Editor for server settings**

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

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the [Information Log Viewer](#). Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.
For network access to file-based databases such as Access or SQL Compact, the **Impersonate** option must be set, so that the TwinCAT Database Server can connect to this network drive. **This feature is currently not supported in Windows CE.**

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

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

**Supported database types**

The installed database types can be selected in the server settings. All installed databases are selected by default. The TwinCAT 3 Database Server will load the corresponding database interfaces. In this way, unused databases on the target system can be deselected.

**Adding a database configuration**

A new database configuration can be added via the command **Add New Database** in the context menu of a Database Server project or via the corresponding command in the toolbar.
A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

**Editor for database configurations**

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

As a further option you can select a so-called failover database (3), which is triggered when an error is encountered in Configure mode. In the event of a network disconnection, this feature can automatically ensure that data are stored elsewhere and not lost.
For each database [122] additional adjustable parameters are available. Depending on the database a connection string (5) is created, which describes the connection to the database. The intention is to make the parameters you have set more transparent.

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

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

**Activating a project**

To activate a configured project on the TwinCAT Database Server, use the command **Activate Configuration** in the context menu of the TwinCAT Database Server project.
The SQL Query Editor is a Database Server tool that supports the development of your application. The tool can be used to test connections and SQL commands and to check the compatibility between PLC and databases.
First select the target system from the routes of your TwinCAT system (1). The TwinCAT Database Server must be installed on the target system. If a NoSQL database is stored in the configuration, an additional NoSQL tab is visible. You will find the documentation in a subitem below.

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

Three commands are available for generating the individual SQL commands:

- Get Table Schema: Calls up the structure of the selected table.
  - Information such as the column name, PLC data type and size of variables is displayed. The retrieved structure can also be prepared for your PLC application via the commands Copy for PLC (4) or Export TC3 (5).

- Create Cmd: An SQL command is generated in the command text box, depending on the selected tab. The command syntax may differ, depending on the database type. The previously read table schema is used here.
  - The created SQL command can optionally be modified.

- Execute: The SQL command shown in the text box is executed and returns values, if applicable.

The differences in the individual SQL commands are explained below.

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

**Create Table command**

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

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

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

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

The Delete command has two functions.

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

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

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

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

The parameter type, name and data type are displayed there. In addition you can insert values here, in order to execute the Stored Procedures with the input values via "Execute". The result is displayed in the output values (4). If several records are returned, the arrow keys can be used to switch between them. This functionality serves as development aid for the call in the PLC. The results are returned there by calling the corresponding function block [198].
InformationLog View for diagnostics

The InformationLog View is available for troubleshooting.

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

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

NoSQL

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

Document-based databases:

Records are stored as documents in the database. This offers the advantage of being able to archive data in a more flexible and hierarchical manner.
If a record consists of more than one flat structure of basic data types, it can no longer be mapped directly via a relational database. NoSQL databases offer this flexibility. Changes in the program code and the corresponding structures can also be easily adopted without having to create a new table.

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

**Build Project**

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

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

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

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

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

**Editor for server settings**

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

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the *Information Log Viewer* [51]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.
For network access to file-based databases such as Access or SQL Compact, the Impersonate option must be set, so that the TwinCAT Database Server can connect to this network drive. This feature is currently not supported in Windows CE.

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

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

**Supported database types**

The installed database types can be selected in the server settings. All installed databases are selected by default. The TwinCAT 3 Database Server will load the corresponding database interfaces. In this way, unused databases on the target system can be deselected.

**Adding a database configuration**

A new database configuration can be added via the command Add New Database in the context menu of a Database Server project or via the corresponding command in the toolbar.
A new database configuration is added in the form of a file in the project folder and integrated in the project. As with all Visual Studio projects, the information on the new files is stored in the Connectivity project.

**Editor for database configurations**

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

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

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

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

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

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

---

**MongoDB in PLC Expert/Configure mode**

PLC Expert and Configure mode use the predefined database schema in their processes. Normally, the schema of the structures used will not change during operation. In order to nevertheless be able to use the function blocks, the TwinCAT 3 Database Server requires a description of the table schema. For MongoDB a table is therefore simulated.

In the SQL Query Editor, use the **SQL** tab and the **CREATE TABLE** subcategory to create a table, or in this case a collection. In addition, unlike for relational databases, an entry is created in a metadata collection. Information on the table schema for the TwinCAT 3 Database Server is stored here.

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

**InformationLog View for diagnostics**

The InformationLog View is available for troubleshooting.

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

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

5.1.2.1 General

5.1.2.1.1 Interface and basic functions

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

The toolbar has the following elements:
<table>
<thead>
<tr>
<th>Toolstrip button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Image" /></td>
<td>Activation of the configuration</td>
</tr>
<tr>
<td><img src="image2" alt="Image" /></td>
<td>Read configuration of the target device</td>
</tr>
<tr>
<td><img src="image3" alt="Image" /></td>
<td>Save configuration in an XML file</td>
</tr>
<tr>
<td><img src="image4" alt="Image" /></td>
<td>Read configuration from an XML file</td>
</tr>
<tr>
<td><img src="image5" alt="Image" /></td>
<td>Add new database configuration</td>
</tr>
<tr>
<td><img src="image6" alt="Image" /></td>
<td>Add new AutoLog group</td>
</tr>
<tr>
<td><img src="image7" alt="Image" /></td>
<td>Event display</td>
</tr>
<tr>
<td><img src="image8" alt="Image" /></td>
<td>Database pool</td>
</tr>
<tr>
<td><img src="image9" alt="Image" /></td>
<td>AutoLog Viewer</td>
</tr>
<tr>
<td><img src="image10" alt="Image" /></td>
<td>InformationLog View</td>
</tr>
<tr>
<td><img src="image11" alt="Image" /></td>
<td>SQL Query Editor</td>
</tr>
</tbody>
</table>
5.1.2.1.2   Project properties

Editor for server settings

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

The settings for logging faults or errors can be configured under **Log settings**. In the event of a fault or error, the Database Server generates a detailed entry in a log file. The log file can be read with the **Information Log Viewer** [P.51]. Under **Log Settings** you can specify a path to the file location and the maximum file size. You can also influence the accuracy of the log. For performance reasons we recommend that logging is deactivated again after the error analysis, once it is no longer required.

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

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

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxStringLength</td>
<td>Maximum string length of the variables in the PLC</td>
</tr>
<tr>
<td>MaxByteArrayLength</td>
<td>Maximum byte array length of the variables in the PLC</td>
</tr>
<tr>
<td>DBNullAllowed</td>
<td>Indicates whether ZERO values are accepted in the TwinCAT Database Server.</td>
</tr>
<tr>
<td>DBCConnectionTimeout</td>
<td>Indicates the time after which the TwinCAT Database Server assumes a connection error while attempts are made to establish a connection.</td>
</tr>
<tr>
<td>DBCCommandTimeout</td>
<td>Indicates the time after which the TwinCAT Database Server assumes a connection fault when a command was sent. If large data quantities are involved, processing of a command may take quite some time, depending on the database and the infrastructure.</td>
</tr>
</tbody>
</table>
Supported database types

The installed database types can be selected in the server settings. All installed databases are selected by default. The TwinCAT 3 Database Server will load the corresponding database interfaces. In this way, unused databases on the target system can be deselected.
5.1.2.1.3 Configuring databases

Editor for database configurations

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

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

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

The CREATE (4) button can be used to create a new database. This function is only displayed if the respective database supports it.
Unknown databases can be configured via an ODBC interface. In the **ODBC Type** drop-down list select "Unknown Database" and add parameters via the commands in the context menu. They may contain passwords, which are stored in encrypted form. The required connection string can be assembled from these parameters. Note that only limited functions of the TwinCAT Database Server can be used. Only the explicit function blocks of the SQL Expert mode are supported.

### 5.1.2.1.4 Configuring AutoLog groups

**Configuring the ADS device**

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

<table>
<thead>
<tr>
<th>ADSDevID:</th>
<th>1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADS Device</td>
<td>local</td>
</tr>
<tr>
<td>AMS NetID</td>
<td>5.19.111.106.1.1</td>
</tr>
<tr>
<td>AMS Port</td>
<td>851</td>
</tr>
<tr>
<td>Timeout</td>
<td>2000</td>
</tr>
<tr>
<td>Connection Type</td>
<td>bySymbolName</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartUp</td>
<td>Manual</td>
</tr>
<tr>
<td>Direction</td>
<td>DeviceAsSource</td>
</tr>
<tr>
<td>Write mode</td>
<td>APPEND</td>
</tr>
<tr>
<td>Ring buffer parameter</td>
<td>0</td>
</tr>
<tr>
<td>Log mode</td>
<td>cyclic</td>
</tr>
<tr>
<td>Cycle Time</td>
<td>500</td>
</tr>
</tbody>
</table>

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

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

**Configuring symbols**

The symbols you set here are written to or read from the database, depending on whether the ADS device is the data target or the data source. The TwinCAT Target browser [P. 52] can be used for convenient access. Here you can search for the symbols on the target and communicate between the two tools via drag & drop.
Symbols can also be added manually to symbol groups or edited. The information that is required varies, depending on whether in the ADS device the connection type was selected via the symbol name or the index groups. The starting point is always the ADS device.
SymbolName | The symbol is addressed based on the set ADS device
Symbol database name | Name of the variable in the database table
DataType | PLC data type of the symbol
BitSize | Bit size of the symbols (set automatically for the data types)
IndexGroup | Index group in the TwinCAT system
IndexOffset | Index offset in the TwinCAT system

Configuring a table

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

The corresponding table can be selected from a list of possible tables. If the table does not yet exist, you can create it via the SQL Query Editor [p. 112]. If you select the standard table structure, a blue tick indicates whether the selected table corresponds to this structure.

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

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

**DB_TO_ADS**

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

**ADS_TO_DB_APPEND**

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

**ADS_TO_DB_UPDATE**

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

**ADS_TO_DB_RINGBUFFER**

This write mode can be used to specify the number of records or the age of records. This write mode is available during cyclic logging via the symbol groups and during logging with the function block FB_DBWrite.

The RingBuffer mode is available for all database types. This mode can also be used to influence logging in ASCII files.
RingBuffer-Arten

The RingBuffer can be used in two different ways:

• "RingBuffer_Time"
• "RingBuffer_Count"

RingBuffer Time

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

RingBuffer Count

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

Declaring the RingBuffer mode in the XML configuration file editor

RingBuffer_Time:

Declaring the RingBuffer mode in FB_DBWrite:

RingBuffer_Time:
**SQL Query Editor**

The SQL Query Editor is a Database Server tool that supports the development of your application. The tool can be used to test connections and SQL commands and to check the compatibility between PLC and databases.
First select the target system from the routes of your TwinCAT system (1). The TwinCAT Database Server must be installed on the target system. If a NoSQL database is stored in the configuration, an additional NoSQL tab is visible. You will find the documentation in a subitem below.

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

Three commands are available for generating the individual SQL commands:

- **Get Table Schema**: Calls up the structure of the selected table.
  - Information such as the column name, PLC data type and size of variables is displayed. The retrieved structure can also be prepared for your PLC application via the commands **Copy for PLC** (4) or **Export TC3** (5).

- **Create Cmd**: An SQL command is generated in the command text box, depending on the selected tab. The command syntax may differ, depending on the database type. The previously read table schema is used here.
  - The created SQL command can optionally be modified.

- **Execute**: The SQL command shown in the text box is executed and returns values, if applicable.

The differences in the individual SQL commands are explained below.

**Select command**

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

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

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

The Delete command has two functions.

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

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

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

The table name can be determined by executing the command. The table with the configured table structure is created.
The TwinCAT Database Server supports "Stored Procedures", which provide numerous databases for processing more complex queries at the database level or to make a simplified interface available.

If Stored Procedures are available in the database and the table, you can list and select them (1). The input and output parameters can be picked up automatically (2) and transferred to the tables in the display (3)(4). The parameter type, name and data type are displayed there. In addition you can insert values here, in order to execute the Stored Procedures with the input values via "Execute". The result is displayed in the output values (4). If several records are returned, the arrow keys can be used to switch between them. This functionality serves as development aid for the call in the PLC. The results are returned there by calling the corresponding function block [198].
NoSql Query Editor

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

Your available databases are then listed in the list of NoSQL databases (3). SQL databases are not displayed in this list. All existing collections are now listed under a selected database. In the menu bar (2) the list can be updated, and collections can be added or deleted.
Once a collection has been selected, the target (6) of the query to be sent changes. The following functions (8) are provided:

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

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

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

Delete: Executes a delete query on the data found with the filter in the text field (7). Any data that is found is deleted from the collection.
**Validate:** If this option is selected, the data queries are not automatically parsed according to their own schema, but an attempt is made to map these data to the structure of the symbol from the PLC, which was specified via these parameters.

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

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

Red: too many or too few data available.

Yellow: The byte length of the record does not match, or underlying records are left over or missing.

Green: No conflicts

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

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

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

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

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

5.1.2.1.7  **InformationLog View**

InformationLog View is a tool for reading log files from the TwinCAT Database Server. Recorded information is displayed with a timestamp, IDs and error messages in plain text.
The log files can not only be viewed or emptied via direct file access, but also directly via the target. This is particularly advantageous with distributed Database Servers in a network, for quick and easy access to the log file. For this access a route to the target device must exist.

### 5.2 Databases

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

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

#### 5.2.1 General Information

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

WSTRING is available for using the Unicode character set.

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

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

<table>
<thead>
<tr>
<th>Database</th>
<th>UTF8</th>
<th>UTF16</th>
<th>Character set definition</th>
<th>Performance impairment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>x</td>
<td>x</td>
<td>column-specific</td>
<td>x</td>
</tr>
<tr>
<td>MSSQL</td>
<td>x</td>
<td>x</td>
<td>column-specific</td>
<td></td>
</tr>
<tr>
<td>Oracle</td>
<td>x</td>
<td>x</td>
<td>column-specific</td>
<td></td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>x</td>
<td>x</td>
<td>cross-database</td>
<td>x</td>
</tr>
<tr>
<td>Others</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Performance impairment**

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

WString support is available from version 3.1.31.4.

### 5.2.1.2 BULK support

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

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

Example command: 'SQLBULK<INSERT>#MyTable'

### 5.2.2 MS SQL database

This section contains information on the configuration and the data type mapping of Microsoft SQL databases.
Compatible versions: Microsoft SQL database 20xx.

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>Microsoft SQL database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Type</strong></td>
</tr>
<tr>
<td><strong>Provider</strong></td>
</tr>
<tr>
<td><strong>Server</strong></td>
</tr>
<tr>
<td><strong>Database</strong></td>
</tr>
<tr>
<td><strong>Authentication</strong></td>
</tr>
<tr>
<td><strong>User name</strong></td>
</tr>
<tr>
<td><strong>Password</strong></td>
</tr>
</tbody>
</table>

Windows CE support

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

Data type mapping between DB and PLC

<table>
<thead>
<tr>
<th>E_ColumnTypes</th>
<th>MS SQL</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>bigint</td>
<td>T_LARGE_INTEGER (TcUtilities.lib)</td>
</tr>
<tr>
<td>Integer</td>
<td>integer</td>
<td>DINT</td>
</tr>
<tr>
<td>SmallInt</td>
<td>smallint</td>
<td>INT</td>
</tr>
<tr>
<td>TinyInt</td>
<td>tinyint</td>
<td>SINT</td>
</tr>
<tr>
<td>Bit_</td>
<td>bit</td>
<td>BYTE</td>
</tr>
<tr>
<td>Money</td>
<td>money</td>
<td>LREAL</td>
</tr>
<tr>
<td>Float</td>
<td>float</td>
<td>LREAL</td>
</tr>
<tr>
<td>Real_</td>
<td>real</td>
<td>REAL</td>
</tr>
<tr>
<td>DateTime</td>
<td>datetime</td>
<td>DT</td>
</tr>
<tr>
<td>NText</td>
<td>ntext</td>
<td>STRING</td>
</tr>
<tr>
<td>NChar</td>
<td>nchar</td>
<td>STRING</td>
</tr>
<tr>
<td>Image</td>
<td>image</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>NVChar</td>
<td>nvarchar</td>
<td>STRING</td>
</tr>
<tr>
<td>Binary</td>
<td>binary</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>VarBinary</td>
<td>varbinary</td>
<td>ARRAY OF BYTE</td>
</tr>
</tbody>
</table>

Data type support

This database supports the data type WSTRING. (See WString support [→123])

NOTE

Data security

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

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.
5.2.2.1 Notes on the Microsoft SQL Server

Logs in the Windows Eventlog

**Error Event**
"Report Server Windows Service (SQLEXPRESS) cannot be connected to the report server database."

In the SQL Configuration Manager under SQL Server 2005 stop the SQL Server Reporting Services (SQLEXPRESS) and set the **Start Mode** to "manual". The Database Server doesn't need the Reporting Service.

**Information event**
"’TcDataLogger’ database is started"

Open the Properties via the context menu in SQL Server Management Studio Express under Databases/TcDataLogger and under Options set the option **Close automatically** to FALSE. This option is not required because the Database Server opens and closes the database automatically.

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

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

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

5.2.3 MS SQL Compact database

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

**Compatible versions:** Microsoft SQL Compact database 3.5

**Declaration in the TwinCAT Database Server Configurator**

<table>
<thead>
<tr>
<th>Microsoft SQL Compact database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Type</strong></td>
<td>Select &quot;Microsoft Compact SQL&quot; from the drop-down menu.</td>
</tr>
<tr>
<td><strong>Database URL</strong></td>
<td>Enter the name and path of the database. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist.</td>
</tr>
<tr>
<td><strong>Authentication</strong></td>
<td>Option for logging into the database with a password.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>Enter the password.</td>
</tr>
</tbody>
</table>

**Windows CE support**

This database is also supported by the Windows CE version of the TwinCAT Database Server. The connection can be established locally.
Data type mapping between DB and PLC

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

Data type support

WSTRING is not supported by this database. (See WString support [p. 123])

Data security

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

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

5.2.4 MySQL database

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

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>MySQL database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Database type</td>
<td>Select &quot;MySQL&quot; from the drop-down menu.</td>
</tr>
<tr>
<td>Server</td>
<td>Enter the name or IP address of your server here.</td>
</tr>
<tr>
<td>Database</td>
<td>Enter the name of the database. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter the port for communicating with the MySQL database here. Default: 3306.</td>
</tr>
<tr>
<td>User name</td>
<td>Enter the user name here.</td>
</tr>
<tr>
<td>Password</td>
<td>Enter the corresponding password.</td>
</tr>
</tbody>
</table>
Windows CE support

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

The installation of the necessary components is not automatic and must be performed manually. To this end, additionally copy version 6.7.8.0 of the library "MySql.Data.CF.dll" into the TwinCAT Database Server folder of the CE device. (\Hard Disk\TwinCAT\Functions\TF6420-Database-Server\Server)

Data type mapping between DB and PLC

<table>
<thead>
<tr>
<th>E_ColumnTypes</th>
<th>MySQL</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>BIGINT</td>
<td>T_ULARGE_INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(TcUtilities.lib)</td>
</tr>
<tr>
<td>Integer</td>
<td>INT</td>
<td>DINT</td>
</tr>
<tr>
<td>SmallInt</td>
<td>SMALLINT</td>
<td>INT</td>
</tr>
<tr>
<td>TinyInt</td>
<td>TINYINT</td>
<td>SINT</td>
</tr>
<tr>
<td>Bit</td>
<td>CHAR(1)</td>
<td>STRING</td>
</tr>
<tr>
<td>Money</td>
<td>DECIMAL(18,4)</td>
<td>LREAL</td>
</tr>
<tr>
<td>Float</td>
<td>DOUBLE</td>
<td>LREAL</td>
</tr>
<tr>
<td>Real_</td>
<td>FLOAT</td>
<td>REAL</td>
</tr>
<tr>
<td>DateTime</td>
<td>DATETIME</td>
<td>DT</td>
</tr>
<tr>
<td>NText</td>
<td>TEXT</td>
<td>STRING</td>
</tr>
<tr>
<td>NChar</td>
<td>CHAR</td>
<td>STRING</td>
</tr>
<tr>
<td>Image</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>NVarChar</td>
<td>VARCHAR</td>
<td>STRING</td>
</tr>
<tr>
<td>Binary</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>VarBinary</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
</tbody>
</table>

Drivers for "big Windows" systems

The used MySQLConnector is under MIT licence and under Copyright:

Copyright (c) 2016 Bradley Grainger

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

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

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

Data type support

This database supports the data type WSTRING. (See WString support [123])
### NOTE

**Data security**

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

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

---

**5.2.5 Oracle database**

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

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

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

---

**Declaration in the TwinCAT Database Server Configurator**

<table>
<thead>
<tr>
<th>Oracle database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Type</strong></td>
</tr>
<tr>
<td><strong>Host</strong></td>
</tr>
<tr>
<td><strong>Service name</strong></td>
</tr>
<tr>
<td><strong>Port</strong></td>
</tr>
<tr>
<td><strong>Protocol</strong></td>
</tr>
<tr>
<td><strong>Scheme</strong></td>
</tr>
<tr>
<td><strong>User name</strong></td>
</tr>
<tr>
<td><strong>Password</strong></td>
</tr>
</tbody>
</table>

---

**Windows CE support**

Under Windows CE this database is not supported by the TwinCAT Database Server.
Data type mapping between DB and PLC

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

● Data type support

This database supports the data type WSTRING. (See WString support [123])

NOTE

Data security

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

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

5.2.6 SQLite

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

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>SQLite database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
</tr>
<tr>
<td>SQLite database file</td>
</tr>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>Password</td>
</tr>
</tbody>
</table>

● Windows CE support

The database is also supported by the Windows CE version of the TwinCAT Database Server, but only on devices with ARM processor. The connection can be established locally.
### Data type mapping between DB and PLC

SQLite has five internal basic data types. For more precise interpretation of the data, additional data types are supported, which are listed in the documentation of the database manufacturer.

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

Special feature: String or binary data types are unlimited in Sqlite. However, the TwinCAT 3 Database Server requires fixed limits, which can be set in the general server settings.

#### Incompatible data types

It is possible that incompatible data type designations are created in the database by third-party software, which the TwinCAT 3 Database Server cannot interpret. In this case it is helpful to use the SQL Query Builder.

#### Data type support

This database supports the data type WSTRING. (See WString support [p. 123])

### Data security

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

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

### 5.2.7 ASCII-File

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

### ASCII as database

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

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

- Data type support
  - WSTRING is not supported by this database. (See [WString support](#123))

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

#### NOTE

Data security
In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.
- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

### 5.2.8 XML database

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

Further information on working with XML files as databases can be found in section "XML - information [► 133]".

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th><strong>XML database</strong></th>
<th>Select &quot;XML&quot; from the drop-down menu.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML Database File</strong></td>
<td>Enter the name and path of the XML file.</td>
</tr>
<tr>
<td><strong>XML Schema File</strong></td>
<td>Enter the name and path of the XSD file.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Enter the name of the database. If the database does not yet exist, it can be created with the Create button. Corresponding permissions must exist. In the case of XML databases, the XML, XSD and XSL files are created automatically.</td>
</tr>
</tbody>
</table>
**Data type support**

WSTRING is not supported by this database. (See WString support [123])

---

**Data type mapping between DB and PLC**

<table>
<thead>
<tr>
<th>E_ColumnTypes</th>
<th>XML</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>bigint</td>
<td>T_LARGE_INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(TcUtilities.lib)</td>
</tr>
<tr>
<td>Integer</td>
<td>integer</td>
<td>DINT</td>
</tr>
<tr>
<td>SmallInt</td>
<td>smallint</td>
<td>INT</td>
</tr>
<tr>
<td>TinyInt</td>
<td>tinyint</td>
<td>BYTE</td>
</tr>
<tr>
<td>Bit</td>
<td>bit</td>
<td>BOOL</td>
</tr>
<tr>
<td>Money</td>
<td>money</td>
<td>LREAL</td>
</tr>
<tr>
<td>Float</td>
<td>float</td>
<td>LREAL</td>
</tr>
<tr>
<td>Real</td>
<td>real</td>
<td>LREAL</td>
</tr>
<tr>
<td>DateTime</td>
<td>datetime</td>
<td>DT</td>
</tr>
<tr>
<td>NText</td>
<td>ntext</td>
<td>STRING</td>
</tr>
<tr>
<td>NChar</td>
<td>nchar</td>
<td>STRING</td>
</tr>
<tr>
<td>Image</td>
<td>image</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>nvarchar</td>
<td>STRING</td>
</tr>
<tr>
<td>Binary</td>
<td>binary</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>VarBinary</td>
<td>varbinary</td>
<td>ARRAY OF BYTE</td>
</tr>
</tbody>
</table>

---

**Data security**

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

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

---

**5.2.8.1 XML - information**

1. Using an XML file as a database with the TwinCAT 3 Database Server
2. Apply XPath queries to an XML file with the TwinCAT 3 Database Server


1. XML as database

**XSD schema for standard table structure:**

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="bigint">
    <xs:restriction base="xs:long" />
  </xs:simpleType>
  <xs:simpleType name="datetime">
    <xs:restriction base="xs:dateTime" />
  </xs:simpleType>
  <xs:simpleType name="ntext_80">
    <xs:restriction base="xs:string" />
  </xs:simpleType>
</xs:schema>
```
XML file for standard table structure (example):

```xml
<?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>
```

Data types for XML tables:

```xml
<xs:simpleTypename="bigint"> //Länge kann individuell angegeben werden
<xs:restrictionbase="xsd:long" />
</xs:simpleTypename>
<xs:simpleTypename="datetime"> //Länge kann individuell angegeben werden
<xs:restrictionbase="xsd:dateTime" />
</xs:simpleTypename>
<xs:simpleTypename="ntext_80"> //Länge kann individuell angegeben werden
<xs:restrictionbase="xsd:string">
   <xs:maxLengthvalue="80" />
</xs:restrictionbase>
</xs:simpleTypename>
<xs:simpleTypename="float">
<xs:restrictionbase="xsd:double" />
</xs:simpleTypename>
<xs:simpleTypename="binary_1"> //Länge kann individuell angegeben werden
<xs:restrictionbase="xsd:hexBinary">
   <xs:maxLengthvalue="1" />
</xs:restrictionbase>
</xs:simpleTypename>
<xs:simpleTypename="bit">
<xs:restrictionbase="xsd:boolean" />
</xs:simpleTypename>
<xs:simpleTypename="image_1"> //Länge kann individuell angegeben werden
<xs:restrictionbase="xsd:hexBinary">
   <xs:maxLengthvalue="1" />
</xs:restrictionbase>
</xs:simpleTypename>
<xs:simpleTypename="integer">
<xs:restrictionbase="xsd:int" />
</xs:simpleTypename>
```
Data type mapping between DB and PLC

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

Creating/reading of records in/from the XML file

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

Samples for supported INSERT commands:

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

Samples for supported SELECT commands:
- 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_DBRewrit
- FB_DBRecordArraySelect
- FB_DBRecordDelete
- FB_DBRecordInsert
- FB_DBRecordInsert_EX
- FB_DBRecordSelect
- FB_DBRecordSelect_EX
- FB_DBTCreate
- FB_DBWrite

2. XML standard XPath function

XPath types

The syntax of the prefixes of the XPaths in the TwinCAT Database Server is as follows:
XPATH_[Type]<[Position]>#Path

There are 4 different types of XPath:
- SEL
  - Reads data from the XML and returns them to the PLC
- ADD
  - Appends the transferred data to the XML at the selected position.
- UPD
  - Replaces the existing XML information at the selected position with the new data.
- DEL
  - Deletes the data in the XML at the selected position.

There are 3 different data available for the positions:
- ATTR
  - Applies to all attribute values from the selected XML tag.
- TAG
  - Applies to the InnerText value of the selected XML tag.
- SUBTAG
  - Applies to the InnerText value of all subtags of the selected XML tag.
  - If an XML Schema exists, the attributes are converted to the correct data types.
  - If no XML Schema exists, the attributes are returned as T_MaxString.
Samples:

**XML file:**

```xml
<?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
<?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 name="Node" maxOccurs="unbounded">
          <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_SEL<ATTR>#TestXML/Node[@attr1=2]

Returned structure if no schema exists:

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

Returned structure if one schema exists:

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

**Sample for XPATH<TAG>**

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

Returned value if no schema exists: SubNode2 : T_MaxString := '200';

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

**Sample for XPATH<SUBTAG>**
XPath => XPATH_SEL<SUBTAG>#TestXML/Node[@attr1=2]

Returned structure if no schema exists:

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

Returned structure if one schema exists:

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

Special feature of the use of the `FB_PLCDBCmd`:

Unlike the usual implementation of the `FB_PLCDBCmd`, the set parameters (ST_ExpParameter) do not specify the placeholders for the respective instruction, but the scheme of the transferred or returned data.

Supported function blocks

- FB_DBRrecordSelect
- FB_DBRrecordSelect_EX
- FB_DBRrecordArraySelect

5.2.9 ODBC databases

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

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

- MySQL
- Oracle
- PostgreSQL
- IBM DB2
- Firebird

Windows CE support

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

5.2.9.1 Free connection string

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

<table>
<thead>
<tr>
<th>ODBC Free Connection String database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>&quot;Odbc_Database&quot;</td>
</tr>
<tr>
<td>ODBC Type</td>
<td>Select &quot;Unknown Database&quot; from the drop-down menu.</td>
</tr>
</tbody>
</table>

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

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

- **Add additional Parameter**
  Adds a general parameter for the connection string. This can have any name, as required by the connection string.

- **Add additional Password Parameter**
  Adds a special password parameter, the value of which is not readable (encrypted) in the configurator and in the configuration file.

**Operating principle with Free Connection String**

To ensure that the TwinCAT Database Server can work with a Free Connection String, the corresponding driver must be installed on the target system of the TwinCAT Database Server. Only "SQL Expert mode [16]" can be used!

**NOTE**

**Data security**

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

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

### 5.2.9.2 MySQL

This section contains information on the configuration and the data type mapping of MySQL databases with ODBC.
Declaration in the TwinCAT Database Server Configurator

**ODBC MySQL database**

<table>
<thead>
<tr>
<th>Database Type</th>
<th>&quot;Odbc_Database&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC Type</td>
<td>Select &quot;MySQL&quot; from the drop-down menu.</td>
</tr>
<tr>
<td>Driver</td>
<td>Enter the actually installed driver.</td>
</tr>
<tr>
<td>Server</td>
<td>Enter the name or IP address of your server.</td>
</tr>
<tr>
<td>Database</td>
<td>Enter the name of the database.</td>
</tr>
<tr>
<td>Port</td>
<td>Enter the port for communicating with the MySQL database here. Default: 3306.</td>
</tr>
<tr>
<td>Option</td>
<td>Default: 2 &quot;Return matched rows instead of affected rows&quot;</td>
</tr>
<tr>
<td>Uid</td>
<td>Enter the user name.</td>
</tr>
<tr>
<td>Pwd</td>
<td>Enter the corresponding password.</td>
</tr>
</tbody>
</table>

**Data type mapping between DB and PLC**

<table>
<thead>
<tr>
<th>E_ColumnTypes</th>
<th>MySQL</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>BIGINT</td>
<td>T_LARGE_INTEGER (TcUtilities.lib)</td>
</tr>
<tr>
<td>Integer</td>
<td>INT</td>
<td>DINT</td>
</tr>
<tr>
<td>SmallInt</td>
<td>SMALLINT</td>
<td>INT</td>
</tr>
<tr>
<td>TinyInt</td>
<td>TINYINT</td>
<td>SINT</td>
</tr>
<tr>
<td>Bit_</td>
<td>CHAR(1)</td>
<td>STRING</td>
</tr>
<tr>
<td>Money</td>
<td>DECIMAL(18,4)</td>
<td>LREAL</td>
</tr>
<tr>
<td>Float</td>
<td>DOUBLE</td>
<td>LREAL</td>
</tr>
<tr>
<td>Real_</td>
<td>FLOAT</td>
<td>REAL</td>
</tr>
<tr>
<td>DateTime</td>
<td>DATETIME</td>
<td>DT</td>
</tr>
<tr>
<td>NText</td>
<td>TEXT</td>
<td>STRING</td>
</tr>
<tr>
<td>NChar</td>
<td>CHAR</td>
<td>STRING</td>
</tr>
<tr>
<td>Image</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>NVarChar</td>
<td>VARCHAR</td>
<td>STRING</td>
</tr>
<tr>
<td>Binary</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>VarBinary</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
</tbody>
</table>

**Data type support**

This database supports the data type WSTRING. (See WString support [123])

**Functioning**

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [138]".

**NOTE**

**Data security**

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

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

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

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>ODBC Oracle database</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>&quot;Odbc_Database&quot;</td>
<td></td>
</tr>
<tr>
<td>ODBC Type</td>
<td>Select &quot;Oracle&quot; from the drop-down menu.</td>
<td></td>
</tr>
<tr>
<td>Driver</td>
<td>Select here the actually installed driver.</td>
<td></td>
</tr>
<tr>
<td>Server</td>
<td>Enter the name or IP address of your server.</td>
<td></td>
</tr>
<tr>
<td>Uid</td>
<td>Enter the user name here.</td>
<td></td>
</tr>
<tr>
<td>Pwd</td>
<td>Enter the corresponding password.</td>
<td></td>
</tr>
</tbody>
</table>

Data type mapping between DB and PLC

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

Data type support

This database supports the data type WSTRING. (See WString support [123])

Functioning

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [138]".

NOTE

Data security

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

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

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

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>ODBC PostgreSQL database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Type</strong></td>
<td>&quot;Odbc_Database&quot;</td>
</tr>
<tr>
<td><strong>ODBC Type</strong></td>
<td>Select &quot;PostgreSQL&quot; from the drop-down menu.</td>
</tr>
<tr>
<td><strong>Driver</strong></td>
<td>Select here the actually installed driver.</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Enter the name or IP address of your server.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Enter the name of the database.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Enter the port for communicating with the PostgreSQL database. Default: 5432.</td>
</tr>
<tr>
<td><strong>Uid</strong></td>
<td>Enter the user name here.</td>
</tr>
<tr>
<td><strong>Pwd</strong></td>
<td>Enter here the corresponding password.</td>
</tr>
</tbody>
</table>

Data type mapping between DB and PLC

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

**Data type support**

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

**Functioning**

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

**NOTE**

Data security

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

- Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.
5.2.9.5 IBM DB2

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

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>ODBC IBM DB2 database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Type</strong></td>
<td>&quot;Odbc_Database&quot;</td>
</tr>
<tr>
<td><strong>ODBC Type</strong></td>
<td>Select &quot;IBM DB2&quot; from the drop-down menu.</td>
</tr>
<tr>
<td><strong>Driver</strong></td>
<td>Enter here the actually installed driver.</td>
</tr>
<tr>
<td><strong>Host name</strong></td>
<td>Enter the name or IP address of your server.</td>
</tr>
<tr>
<td><strong>Database</strong></td>
<td>Enter the name of the database.</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>Enter the port for communicating with the IBM DB2 database. Default: 50000.</td>
</tr>
<tr>
<td><strong>Protocol</strong></td>
<td>Default: TCPIP</td>
</tr>
<tr>
<td><strong>Uid</strong></td>
<td>Enter the user name here.</td>
</tr>
<tr>
<td><strong>Pwd</strong></td>
<td>Enter here the corresponding password.</td>
</tr>
<tr>
<td><strong>LONGDATACOMPAT</strong></td>
<td>Default: 1</td>
</tr>
</tbody>
</table>

Data type mapping between DB and PLC

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

- **Data type support**
  WSTRING is not supported by this database. (See WString support [123])

- **Functioning**
  All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [138]".
NOTE

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

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

5.2.9.6 Firebird

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

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>ODBC Firebird database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Type</td>
<td>&quot;Odbc_Database&quot;</td>
</tr>
<tr>
<td>ODBC Type</td>
<td>Select &quot;Firebird&quot; from the drop-down menu.</td>
</tr>
<tr>
<td>Driver</td>
<td>Select here the actually installed driver.</td>
</tr>
<tr>
<td>Database</td>
<td>Enter the name of the database.</td>
</tr>
<tr>
<td>Client</td>
<td></td>
</tr>
<tr>
<td>Uid</td>
<td>Enter the user name here.</td>
</tr>
<tr>
<td>Pwd</td>
<td>Enter the corresponding password.</td>
</tr>
</tbody>
</table>

Data type mapping between DB and PLC

<table>
<thead>
<tr>
<th>E_ColumnType</th>
<th>Firebird</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_BigInt</td>
<td>BIGINT</td>
<td>T_ULARGE_INTEGER (TcUtilities.lib)</td>
</tr>
<tr>
<td>Integer</td>
<td>INTEGER</td>
<td>DINT</td>
</tr>
<tr>
<td>SmallInt</td>
<td>SMALLINT</td>
<td>INT</td>
</tr>
<tr>
<td>TinyInt</td>
<td>TINYINT</td>
<td>INT</td>
</tr>
<tr>
<td>Bit</td>
<td>CHAR(1)</td>
<td>STRING</td>
</tr>
<tr>
<td>Money</td>
<td>DECIMAL(18,4)</td>
<td>LREAL</td>
</tr>
<tr>
<td>Float</td>
<td>FLOAT</td>
<td>REAL</td>
</tr>
<tr>
<td>Real_</td>
<td>DOUBLE PRECISION</td>
<td>LREAL</td>
</tr>
<tr>
<td>DateTime</td>
<td>TIMESTAMP</td>
<td>DT</td>
</tr>
<tr>
<td>NText</td>
<td>VARCHAR(254)</td>
<td>STRING</td>
</tr>
<tr>
<td>NChar</td>
<td>CHAR(254)</td>
<td>STRING</td>
</tr>
<tr>
<td>Image</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>NVChar</td>
<td>VARCHAR(254)</td>
<td>STRING</td>
</tr>
<tr>
<td>Binary</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>VarBinary</td>
<td>BLOB</td>
<td>ARRAY OF BYTE</td>
</tr>
</tbody>
</table>

Data type support

WSTRING is not supported by this database. (See WString support [123])

Functioning

All functions of the TwinCAT Database Server can be applied to the ODBC templates. This does not apply to the "Free Connection String [138]".
**NOTE**

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

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

---

### 5.2.10 MS Access database

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

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

#### Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th><strong>Microsoft Access database</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBValueType</strong></td>
<td>Select &quot;Double&quot; to limit logging to alphanumeric and Boolean data types. Select &quot;Bytes&quot; to also log structures and strings.</td>
</tr>
<tr>
<td><strong>DBType</strong></td>
<td>Select &quot;MS Access&quot;. PLC: eDBType_Access.</td>
</tr>
<tr>
<td><strong>DBServer</strong></td>
<td>Not required.</td>
</tr>
<tr>
<td><strong>DBUrl</strong></td>
<td>DBUrl contains the path to the MDB file. e.g. <code>C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.mdb</code></td>
</tr>
<tr>
<td><strong>DBTable</strong></td>
<td>DBTable contains the name of the table.</td>
</tr>
</tbody>
</table>

#### Windows CE support

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

---

### Data type mapping between DB and PLC

<table>
<thead>
<tr>
<th><strong>E_DBColumnTypes</strong></th>
<th><strong>MS Access</strong></th>
<th><strong>PLC Control</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>eDBCOLUMN_BigInt</td>
<td>Integer4</td>
<td>DINT</td>
</tr>
<tr>
<td>eDBCOLUMN_Integer</td>
<td>Integer2</td>
<td>INT</td>
</tr>
<tr>
<td>eDBCOLUMN_SmallInt</td>
<td>Integer2</td>
<td>SINT</td>
</tr>
<tr>
<td>eDBCOLUMN_TinyInt</td>
<td>Integer1</td>
<td>SINT</td>
</tr>
<tr>
<td>eDBCOLUMN_Bit</td>
<td>YESNO</td>
<td>BYTE</td>
</tr>
<tr>
<td>eDBCOLUMN_Money</td>
<td>Currency</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_Float</td>
<td>Double</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_Real</td>
<td>Single</td>
<td>REAL</td>
</tr>
<tr>
<td>eDBCOLUMN_DateTime</td>
<td>DATETIME</td>
<td>DT</td>
</tr>
<tr>
<td>eDBCOLUMN_NText</td>
<td>Text</td>
<td>STRING</td>
</tr>
<tr>
<td>eDBCOLUMN_NChar</td>
<td>VarChar</td>
<td>STRING</td>
</tr>
<tr>
<td>eDBCOLUMN_Image</td>
<td>OLEOBJECT</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>eDBCOLUMN_NVarChar</td>
<td>VarChar</td>
<td>STRING</td>
</tr>
<tr>
<td>eDBCOLUMN_Binary</td>
<td>OLEOBJECT</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>eDBCOLUMN_VarBinary</td>
<td>OLEOBJECT</td>
<td>ARRAY OF BYTE</td>
</tr>
</tbody>
</table>
Data type support
WSTRING is not supported by this database. (See WString support [123])

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

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

5.2.11 MS Excel database
The variable values are stored in an Microsoft Excel database.

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>Microsoft Excel database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DBValueType</strong></td>
<td>Select &quot;Double&quot; to limit logging to alphanumeric and Boolean data types. Select &quot;Bytes&quot; to also log structures and strings.</td>
</tr>
<tr>
<td><strong>DBType</strong></td>
<td>Select &quot;MS Excel&quot;. PLC: eDBType_MSExcel.</td>
</tr>
<tr>
<td><strong>DBServer</strong></td>
<td>Not required.</td>
</tr>
<tr>
<td><strong>DBProvider</strong></td>
<td>&quot;Microsoft.OLEDB.4.0&quot; or &quot;Microsoft.ACE.OLEDB.12.0&quot;</td>
</tr>
<tr>
<td><strong>DBUrl</strong></td>
<td>DBUrl contains the path to the Excel file.</td>
</tr>
<tr>
<td></td>
<td>e.g. C:\TwinCAT\TcDatabaseSrv\Samples\TestDB.xls</td>
</tr>
<tr>
<td><strong>DBTable</strong></td>
<td>DBTable contains the name of the table.</td>
</tr>
</tbody>
</table>

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

Data type mapping between DB and PLC

<table>
<thead>
<tr>
<th><em>E_DBColumnTypes</em></th>
<th>MS Excel</th>
<th>PLC Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>eDBCOLUMN_BigInt</td>
<td>Number</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_Integer</td>
<td>Number</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_SmallInt</td>
<td>Number</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_TinyInt</td>
<td>Number</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_Bit</td>
<td>BOOLEAN</td>
<td>BOOL</td>
</tr>
<tr>
<td>eDBCOLUMN_Money</td>
<td>Currency</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_Float</td>
<td>Number</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_Real</td>
<td>Number</td>
<td>LREAL</td>
</tr>
<tr>
<td>eDBCOLUMN_DateTime</td>
<td>Date</td>
<td>DT</td>
</tr>
<tr>
<td>eDBCOLUMN_NText</td>
<td>Text</td>
<td>STRING(255)</td>
</tr>
<tr>
<td>eDBCOLUMN_NChar</td>
<td>Text</td>
<td>STRING(255)</td>
</tr>
<tr>
<td>eDBCOLUMN_NVarChar</td>
<td>Text</td>
<td>STRING(255)</td>
</tr>
</tbody>
</table>

Functions that are not supported
Automatic ID generation is not supported by this database. If the standard table structure is used in Configure mode, the value of the ID is not set.
Non-supported data types
Binary, VarBinary and Image are not supported with Excel databases.

Data type support
WSTRING is not supported by this database. (See WString support [123])

NOTE
Data security
In flash memory devices the number of write access operations is limited. The flash memory devices can fail, with a risk of data loss.
• Make regular backups of your system. Use the IPC diagnostics in order to determine the status of the flash memory devices.

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

Declaration in the TwinCAT Database Server Configurator

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

Windows CE support
Under Windows CE this database is not supported by the TwinCAT Database Server.
Data type mapping between DB and PLC

<table>
<thead>
<tr>
<th>MongoDB</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>LINT</td>
</tr>
<tr>
<td>int</td>
<td>DINT</td>
</tr>
<tr>
<td>bool</td>
<td>BYTE</td>
</tr>
<tr>
<td>double</td>
<td>LREAL</td>
</tr>
<tr>
<td>timestamp</td>
<td>DT</td>
</tr>
<tr>
<td>string</td>
<td>STRING</td>
</tr>
<tr>
<td>binData</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>objectId</td>
<td>T_ObjectId_MongoDB</td>
</tr>
<tr>
<td>array</td>
<td>ARRAY</td>
</tr>
<tr>
<td>object</td>
<td>STRUCT</td>
</tr>
</tbody>
</table>

Data type support

WSTRING is not supported by this database. (See WString support [123])

MongoDB in PLC Expert/Configure mode

PLC Expert and Configure mode use the predefined database schema in their processes. Normally, the schema of the structures used will not change during operation. In order to nevertheless be able to use the function blocks, the TwinCAT 3 Database Server requires a description of the table schema. For MongoDB a table is therefore simulated.

In the SQL Query Editor, use the SQL tab and the CREATE TABLE subcategory to create a table, or in this case a collection. In addition, unlike for relational databases, an entry is created in a metadata collection. Information on the table schema for the TwinCAT 3 Database Server is stored here.

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

Use of certificates

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

<table>
<thead>
<tr>
<th>User name</th>
<th>User name of the corresponding certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate Authority</td>
<td>Path to the SSL certificate of the certificate authority. This may be a self-signed certificate.</td>
</tr>
<tr>
<td>Client Certificate</td>
<td>Client certificate signed by the SSL certificate.</td>
</tr>
<tr>
<td>Client Certificate Password</td>
<td>Password of the client certificate.</td>
</tr>
</tbody>
</table>

Configuring the database connection to MongoDB using certificates:
NOTE

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

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

5.2.13 PostgreSQL

This section provides information about configuration and application of the PostgreSql database. PostgreSQL is an object-relational open source database with a client-server infrastructure. The TwinCAT 3 database server uses the Npgsql API for connection.

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>PostgreSQL database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database type</td>
</tr>
<tr>
<td>Server</td>
</tr>
<tr>
<td>Database</td>
</tr>
<tr>
<td>Port</td>
</tr>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>User name</td>
</tr>
<tr>
<td>Password</td>
</tr>
<tr>
<td>Client Certificate</td>
</tr>
<tr>
<td>Client Certificate Password</td>
</tr>
</tbody>
</table>

- Windows CE support
  Under Windows CE this database is not supported by the TwinCAT Database Server.
Data type mapping between DB and PLC

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

**NOTE**

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

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

**Data type support**

WSTRING is not supported by this database. (See WString support [123])

5.2.14 InfluxDB

This section contains information about the configuration and the data type mapping of InfluxDB.

Supported version: 1.7.x, 1.8.x

Declaration in the TwinCAT Database Server Configurator

<table>
<thead>
<tr>
<th>InfluxDB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Database type</td>
<td>Select &quot;InfluxDB&quot; from the drop-down menu.</td>
</tr>
<tr>
<td>Server</td>
<td>Enter the address of the desired database server here.</td>
</tr>
<tr>
<td>Database</td>
<td>Enter the name of the database. If the database does not yet exist, you can create it with &quot;Create&quot;.</td>
</tr>
<tr>
<td>Authentication</td>
<td>None: No authentication</td>
</tr>
<tr>
<td></td>
<td>User name/password: Login with user name and password</td>
</tr>
</tbody>
</table>

**Windows CE support**

Under Windows CE this database is not supported by the TwinCAT Database Server.
**Data type mapping between DB and PLC**

<table>
<thead>
<tr>
<th>E_ColumnTypes</th>
<th>InfluxDB</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInt</td>
<td>Integer</td>
<td>LINT</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer</td>
<td>DINT</td>
</tr>
<tr>
<td>SmallInt</td>
<td>Integer</td>
<td>INT</td>
</tr>
<tr>
<td>TinyInt</td>
<td>Integer</td>
<td>BYTE</td>
</tr>
<tr>
<td>Bit</td>
<td>Boolean</td>
<td>BOOL</td>
</tr>
<tr>
<td>Money</td>
<td>Float</td>
<td>LREAL</td>
</tr>
<tr>
<td>Float</td>
<td>Float</td>
<td>LREAL</td>
</tr>
<tr>
<td>Real</td>
<td>Float</td>
<td>LREAL</td>
</tr>
<tr>
<td>DateTime</td>
<td>Integer</td>
<td>DT</td>
</tr>
<tr>
<td>NText</td>
<td>String</td>
<td>STRING</td>
</tr>
<tr>
<td>NChar</td>
<td>String</td>
<td>STRING</td>
</tr>
<tr>
<td>Image</td>
<td>String</td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>NVarChar</td>
<td>String</td>
<td>STRING</td>
</tr>
<tr>
<td>Binary</td>
<td></td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>VarBinary</td>
<td></td>
<td>ARRAY OF BYTE</td>
</tr>
<tr>
<td>Tag</td>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>&quot;time&quot;</td>
<td></td>
<td>LINT</td>
</tr>
</tbody>
</table>

**Time**

As a time series database, InfluxDB has some special features. Each measurement (table) of a series contains the time column, tag columns and field columns. The time column is stored in the database as UNIX epoch time. The function blocks of the database server use the TwinCAT time (number of 100 ns steps since January 1, 1601). They are converted to UNIX epoch time. The FB_SQLDBCommand is excluded from this conversion. Here you can transfer your own free timestamps without conversion. The precision is set to "ns" accuracy. Times are unique in InfluxDB as ID together with the tag columns. If the tags and the time are identical, a data set is overwritten.

**Standard table structure**

The standard table structure for InfluxDB looks like this in the PLC:

<table>
<thead>
<tr>
<th>ColumnName</th>
<th>InfluxDB</th>
<th>TwinCAT PLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Integer</td>
<td>LINT</td>
</tr>
<tr>
<td>Name</td>
<td>Tag</td>
<td>T_MaxString</td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
<td>LREAL</td>
</tr>
</tbody>
</table>

**Data type support**

WSTRING is not supported by this database. (See WString support [123])

**NOTE**

**Data security**

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

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

6.1  Tc3_Database

6.1.1  Function blocks

The function blocks of the Tc3_Database compiled library are split into three sections, based on the basic concept:

- **Configure Mode:**
  Contains function blocks for controlling reading and writing of AutoLog groups defined in the configurator.

- **PLC Expert Mode:**
  Contains function blocks for conventional PLC programmers.

- **SQL Expert mode:**
  IT and PLC experts with advanced database knowledge can use these function blocks to assemble SQL commands in the PLC.

- **NoSql Expert Mode:**
  These function blocks can be used by IT and PLC experts with extended database knowledge to create commands via NoSQL databases and send them to the database.

**Using the Tc3_Eventlogger**

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

6.1.1.1  Configure mode

6.1.1.1.1  FB_ConfigTcDBSrvEvt

![FB_ConfigTcDBSrvEvt](image)

Function block for creating, reading and deleting configuration entries for the TwinCAT Database Server.

**Syntax**

**Definition:**

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

Version: 1.11
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Local</td>
<td>Creates new entries in the XML configuration file for the TwinCAT Database Server</td>
</tr>
<tr>
<td>Read</td>
<td>Local</td>
<td>Reads the current configuration of the TwinCAT Database Server</td>
</tr>
<tr>
<td>Delete</td>
<td>Local</td>
<td>Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server</td>
</tr>
</tbody>
</table>

## Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

## Create

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

### Syntax

```plaintext
METHOD Create : BOOL
VAR_INPUT
    pTcDbsrvConfig: POINTER TO BYTE;
    cbTcDbsrvConfig: UDINT;
    bTemporary: BOOL := TRUE;
    pConfigID: POINTER TO UDINT;
END_VAR
```
**Inputs**

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

Creating AutoLog groups is currently not supported.

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

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

**Read**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDBConfig</td>
<td>POINTER TO ARRAY [1..MAX CONFIGURATIONS] OF ST_ConfigDB</td>
<td>Pointer address of the array into which the database configurations are to be written.</td>
</tr>
<tr>
<td>cbDBConfig</td>
<td>UDINT</td>
<td>Length of the database configuration array</td>
</tr>
<tr>
<td>pAutoLogGrpConfig</td>
<td>POINTER TO ARRAY[1..MAX CONFIGURATIONS] OF ST_ConfigAutoLogGrp</td>
<td>Pointer address of the array into which the AutoLogGrp configurations are to be written.</td>
</tr>
<tr>
<td>cbAutoLogGrpConfig</td>
<td>UDINT</td>
<td>Length of the AutoLogGrp configuration array</td>
</tr>
<tr>
<td>pDBCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of database configurations.</td>
</tr>
<tr>
<td>pAutoLogGrpCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of AutoLogGrp configurations.</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

## Sample

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

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

## Delete

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

## Syntax

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTcDBSrvConfigType</td>
<td>E_TcDBSrvConfigType</td>
<td>Type of the configuration to be deleted (database / AutoLog group)</td>
</tr>
<tr>
<td>hConfigID</td>
<td>UDINT</td>
<td>ID of the configuration to be deleted (hDBID or hAutoLogGrpID)</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

## Sample

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

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

6.1.1.1.2 FB_PLCDBAutoLogEvt

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

### Syntax

**Definition:**

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

```plaintext
FB_PLCDBAutoLogEvt

<table>
<thead>
<tr>
<th>sNetID</th>
<th>_AmsNetID</th>
<th>BOOL</th>
<th>bBusy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout</td>
<td>TIME</td>
<td></td>
<td>bError</td>
</tr>
<tr>
<td></td>
<td>I_TcMessage</td>
<td>ipTcResult</td>
<td>bBusy_Status</td>
</tr>
</tbody>
</table>
```
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

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

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>Local</td>
<td>Executes the AutoLog group once</td>
<td>[157]</td>
</tr>
<tr>
<td>Start</td>
<td>Local</td>
<td>Starts AutoLog mode with the corresponding configured AutoLog groups</td>
<td>[158]</td>
</tr>
<tr>
<td>Status</td>
<td>Local</td>
<td>Queries the status of the AutoLog groups.</td>
<td>[158]</td>
</tr>
<tr>
<td>Stop</td>
<td>Local</td>
<td>Stops AutoLog mode</td>
<td>[159]</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**RunOnce**

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

**Syntax**

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

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAutoLogGrpID</td>
<td>UDINT</td>
<td>ID of the AutoLog group to be executed once.</td>
</tr>
<tr>
<td>bAll</td>
<td>BOOL</td>
<td>If TRUE, all AutoLog groups are executed once.</td>
</tr>
</tbody>
</table>
Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

Start

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

Syntax

```plaintext
METHOD Start : BOOL
```

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

Status

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

Syntax

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCheckCycle</td>
<td>TIME</td>
<td>Interval time at which the status array is updated.</td>
</tr>
<tr>
<td>pError</td>
<td>POINTER TO BOOL</td>
<td>TRUE, if an error has occurred in AutoLog mode.</td>
</tr>
<tr>
<td>pAutoLogStatus</td>
<td>POINTER TO ARRAY [1..MAXCONFIGURATIONS][243] OF ST_AutoLogGrpStatus [240]</td>
<td>Address of the status array that contains all groups.</td>
</tr>
<tr>
<td>cbAutoLogStatus</td>
<td>UDINT</td>
<td>Length of the status array</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

Stop

This method stops the AutoLog mode.

Syntax

```
METHOD Stop : BOOL
```

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

```
VAR
    fbPLCDBAutoLog : FB_PLCDBAutoLogEvt (sNetID='');
END_VAR

IF fbPLCDBAutoLog.Stop() THEN
    // ...
END_IF
```
6.1.1.2 PLC Expert mode

PLC controlled Autolog Mode usecase

Independent functionsblocks of the PLC Expert Mode

6.1.1.2.1 FB_ConfigTcDBSrvEvt

Syntax

Function block for creating, reading and deleting configuration entries for the TwinCAT Database Server.

Definition:

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage[221]</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity[222]</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Local</td>
<td>Creates new entries in the XML configuration file for the TwinCAT Database Server</td>
</tr>
<tr>
<td>Read</td>
<td>Local</td>
<td>Reads the current configuration of the TwinCAT Database Server</td>
</tr>
<tr>
<td>Delete</td>
<td>Local</td>
<td>Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Create**

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

**Syntax**

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

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

Creating AutoLog groups is currently not supported.

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

VAR

```plaintext
VAR
  fbConfigTcDBSrv : FB_ConfigTcDBSrvEvt(sNetId := '', tTimeout:=T#5S);
  myConfigHandle : INT;
  stConfigDB : T_DBConfig MsCompactSQL;
  tcMessage : _TcMessage;
END_VAR
```

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

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

Read

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

Syntax

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDBConfig</td>
<td>POINTER TO ARRAY [1..MAX CONFIGURATIONS] OF ST_ConfigDB</td>
<td>Pointer address of the array into which the database configurations are to be written.</td>
</tr>
<tr>
<td>cbDBConfig</td>
<td>UDINT</td>
<td>Length of the database configuration array</td>
</tr>
<tr>
<td>pAutoLogGrpConfig</td>
<td>POINTER TO ARRAY [1..MAX CONFIGURATIONS] OF ST_ConfigAutoLogGrp</td>
<td>Pointer address of the array into which the AutoLogGrp configurations are to be written.</td>
</tr>
<tr>
<td>cbAutoLogGrpConfig</td>
<td>UDINT</td>
<td>Length of the AutoLogGrp configuration array</td>
</tr>
<tr>
<td>pDBCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of database configurations.</td>
</tr>
<tr>
<td>pAutoLogGrpCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of AutoLogGrp configurations.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

**Delete**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTcDBSrvConfigType</td>
<td>E_TcDBSrvConfigType</td>
<td>Type of the configuration to be deleted (database / AutoLog group)</td>
</tr>
<tr>
<td>hConfigID</td>
<td>UDINT</td>
<td>ID of the configuration to be deleted (hDBID or hAutoLogGrpID)</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

### 6.1.1.2.2 FB_PLCDBAutoLogEvt

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

**Syntax**

**Definition:**

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

## Outputs

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

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>Local</td>
<td>Executes the AutoLog group once</td>
</tr>
<tr>
<td>Start</td>
<td>Local</td>
<td>Starts AutoLog mode with the corresponding configured AutoLog groups</td>
</tr>
<tr>
<td>Status</td>
<td>Local</td>
<td>Queries the status of the AutoLog groups.</td>
</tr>
<tr>
<td>Stop</td>
<td>Local</td>
<td>Stops AutoLog mode</td>
</tr>
</tbody>
</table>

## Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

## RunOnce

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

### Syntax

```plaintext
METHOD RunOnce : BOOL
VAR_INPUT
  hAutoLogGrpID: UDINT;
  bAll: BOOL;
END_VAR
```
**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

**Start**

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

**Syntax**

```plaintext
METHOD Start : BOOL
```

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

**Status**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCheckCycle</td>
<td>TIME</td>
<td>Interval time at which the status array is updated.</td>
</tr>
<tr>
<td>pError</td>
<td>POINTER TO BOOL</td>
<td>TRUE, if an error has occurred in AutoLog mode.</td>
</tr>
<tr>
<td>pAutoLogStatus</td>
<td>POINTER TO ARRAY</td>
<td>Address of the status array that contains all groups.</td>
</tr>
<tr>
<td>cbAutoLogStatus</td>
<td>UDINT</td>
<td>Length of the status array</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

### Sample

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

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

### Stop

This method stops the AutoLog mode.

### Syntax

```plaintext
METHOD Stop : BOOL
```

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

### Sample

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

IF fbPLCDBAutoLog.Stop() THEN
  ; // ...
END_IF
6.1.1.2.3 FB_PLCDBCreateEvt

Function block with two methods. One method can be used to create databases from the PLC on a database server specified in the PLC. The other method can be used to generate a new table in a specified database.

**Syntax**

Definition:

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

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage [221]</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Properties</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSev [222]</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Local</td>
<td>Creates a new database</td>
</tr>
<tr>
<td>Table</td>
<td>Local</td>
<td>Creates a new table with a structure that is defined via an array with x elements or x columns in the PLC.</td>
</tr>
</tbody>
</table>
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Database

This method creates a new database. Optionally you can specify whether the created database should also be used for the configuration of the TwinCAT Database Server.

Syntax

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

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDatabaseConfig</td>
<td>POINTER TO BYTE</td>
<td>Address of the database configuration structure</td>
</tr>
<tr>
<td>cbDatabaseConfig</td>
<td>UDINT</td>
<td>Length of the database configuration structure</td>
</tr>
<tr>
<td>bCreateXMLConfig</td>
<td>BOOL</td>
<td>Indicates whether the newly created database should be entered as new configuration entry in the XML file.</td>
</tr>
<tr>
<td>pDBID</td>
<td>UDINT</td>
<td>Returns the hDBID if/when a new configuration entry was created.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

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

This method creates a new table with a structure that is defined through an array with \( x \) elements or \( x \) columns in the PLC.

Syntax

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

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>MaxString</td>
<td>Name of the table to be created.</td>
</tr>
<tr>
<td>pTableCfg</td>
<td>POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ColumnInfo</td>
<td>Indicates the pointer address of the table structure array. The individual columns are written in this array.</td>
</tr>
<tr>
<td>cbTableCfg</td>
<td>UDINT</td>
<td>Indicates the length of the array in which the columns are configured.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

ColumnInfo[0].sName := 'colBigInt'; ColumnInfo[0].eType := E_ColumnType.BigInt; ColumnInfo[0].nLength := 8;
ColumnInfo[1].sName := 'colInteger'; ColumnInfo[1].eType := E_ColumnType.Integer; ColumnInfo[1].nLength := 4;
ColumnInfo[2].sName := 'colSmallInt'; ColumnInfo[2].eType := E_ColumnType.SmallInt; ColumnInfo[2].nLength := 2;
ColumnInfo[3].sName := 'colTinyInt'; ColumnInfo[3].eType := E_ColumnType.TinyInt; ColumnInfo[3].nLength := 1;
ColumnInfo[4].sName := 'colBit'; ColumnInfo[4].eType := E_ColumnType.BIT_; ColumnInfo[4].nLength := 1;
ColumnInfo[5].sName := 'colMoney'; ColumnInfo[5].eType := E_ColumnType.Money; ColumnInfo[5].nLength := 8;
ColumnInfo[6].sName := 'colFloat'; ColumnInfo[6].eType := E_ColumnType.Float; ColumnInfo[6].nLength := 8;
ColumnInfo[7].sName := 'colReal'; ColumnInfo[7].eType := E_ColumnType.REAL_; ColumnInfo[7].nLength := 4;
ColumnInfo[8].sName := 'colDateTime'; ColumnInfo[8].eType := E_ColumnType.DateTime; ColumnInfo[8].nLength := 4;
ColumnInfo[9].sName := 'colText'; ColumnInfo[9].eType := E_ColumnType.Text; ColumnInfo[9].nLength := 256;
ColumnInfo[10].sName := 'colNChar'; ColumnInfo[10].eType := E_ColumnType.NChar; ColumnInfo[10].nLength := 10;
ColumnInfo[12].sName := 'colNVarChar'; ColumnInfo[12].eType := E_ColumnType.NVarChar; ColumnInfo[12].nLength := 50;
ColumnInfo[13].sName := 'colBinary'; ColumnInfo[13].eType := E_ColumnType.Binary; ColumnInfo[13].nLength := 256;
3].nLength := 30;
ColumnInfo[14].sName := 'colVarBinary'; ColumnInfo[14].eType := E_ColumnType.VarBinary; ColumnInfo[14].nLength := 20;

IF fbPLCDBCreate.Table(
    hDBID:= 1,
    sTableName:= 'myNewTable',
    pTableCfg:= ADR(ColumnInfo),
    cbTableCfg:= SIZEOF(ColumnInfo))
THEN
    IF fbPLCDBCreate.bError THEN
        TcMessage:= fbPLCDBCreate.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
ELSE
END_IF

6.1.1.2.4 FB_PLCDBReadEvt

Function block for reading records from a database.

**Syntax**

**Definition:**

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

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRecords</td>
<td>UDINT</td>
<td>Outputs the maximum number of records that could be collected depending on sDBSymbolName.</td>
</tr>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Local</td>
<td>Reads a specified number of records from a database table with the standard table structure specified by Beckhoff.</td>
</tr>
<tr>
<td>ReadStruct</td>
<td>Local</td>
<td>Reads a specified number of records from a database table with any table structure.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Read

This method reads a specified number of records from a database table with the standard table structure specified by Beckhoff. The standard table structure is used in AutoLog mode and in the FB_DBWriteEvt function block, for example.

Syntax

METHOD Read : BOOL
VAR_INPUT
　hDBID: UDINT;
　sTableName: T_MaxString;
　sDBSymbolName: T_MaxString;
　eOrderBy: E_OrderColumn := E_OrderColumn.eColumnID;
　eOrderType: E_OrderType := E_OrderType.eOrder_ASC;
　nStartIndex: UDINT;
　nRecordCount: UDINT;
　pData: POINTER TO ST_StandardRecord;
　cbData: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>sDBSymbolName</td>
<td>T_MaxString</td>
<td>Symbol name to be read from the standard table structure.</td>
</tr>
<tr>
<td>eOrderBy</td>
<td>E_OrderColumn.eColumnID</td>
<td>Sorting column (ID, timestamp, name or value)</td>
</tr>
<tr>
<td>eOrderType</td>
<td>E_OrderType.eOrder_ASC</td>
<td>Sorting direction (ASC or DESC)</td>
</tr>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO ST_StandardRecord</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
</tbody>
</table>
Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

```plaintext
VAR
    fbPLCDBRead : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
    tcMessage   : I_TcMessage;
END_VAR

IF fbPLCDBRead.Read(
    hDBID:= 1,
    sTableName:= 'MyTable_WithLReal',
    sDBSymbolName:= 'MyValue',
    eOrderBy:= E_OrderColumn.ID,
    eOrderType:= E_OrderType.DESC,
    nStartIndex:= 0,
    nRecordCount:= 1,
    pData:= ADR(ReadStruct),
    cbData:= SIZEOF(ReadStruct))
THEN
    IF fbPLCDBRead.bError THEN
        tcMessage := fbPLCDBRead.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

Result in the PLC:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadStruct</td>
<td>ST_StandardRecord</td>
<td></td>
</tr>
<tr>
<td>nID</td>
<td>UINT</td>
<td>2</td>
</tr>
<tr>
<td>dtTimestamp</td>
<td>DATE_AND_TIME</td>
<td>DT#2018-2-1-16:8:8</td>
</tr>
<tr>
<td>sName</td>
<td>STRING(80)</td>
<td>'MyValue'</td>
</tr>
<tr>
<td>vValue</td>
<td>REAL</td>
<td>15.9</td>
</tr>
</tbody>
</table>

ReadStruct

This method reads a specified number of records from a database table with any table structure.

Syntax

```plaintext
METHOD ReadStruct : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
    cbColumnNames: UDINT;
    sOrderByColumn: STRING(50);
    eOrderType: E_OrderType := E_OrderType.eOrder_ASC
    nStartIndex: UDINT;
    nRecordCount: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
END_VAR
```
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>pColumnNames</td>
<td>POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)</td>
<td>Address of the array containing the column name to be read.</td>
</tr>
<tr>
<td>cbColumnNames</td>
<td>UDINT</td>
<td>Length of the column name array</td>
</tr>
<tr>
<td>sOrderByColumn</td>
<td>STRING(50)</td>
<td>Name the sorting column</td>
</tr>
<tr>
<td>eOrderType</td>
<td>E_OrderType</td>
<td>Sorting direction (ASC or DESC)</td>
</tr>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadStruct</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

### Sample

```plaintext
VAR
  fbPLCDBRead : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
  myCustomStruct : ST_Record;
  tcMessage : I_TcMessage;
END_VAR

TYPE ST_Record :
  STRUCT
    nID : LINT;
    dtTimestamp : DATE_AND_TIME;
    sName : STRING;
    nSensor1 : LREAL;
    nSensor2 : LREAL;
  END_STRUCT
END_TYPE

// set column names
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';

IF fbPLCDBRead.ReadStruct(
  hDBID:= 1,
  sTableName:= 'MyTable_Struct',
  pColumnNames:= ADR(ColumnNames),
  cbColumnNames:= SIZEOF(ColumnNames),
  sOrderByColumn:= ColumnNames[0],
  eOrderType:= E_OrderType.DESC,
  nStartIndex:= 0,
  nRecordCount:= 1,
  pData:= ADR(myCustomStruct),
  cbData:= SIZEOF(myCustomStruct))
THEN
  IF fbPLCDBRead.bError THEN
    tcMessage:= fbPLCDBRead.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF
```

### Result in the PLC:
6.1.1.2.5 FB_PLCDBWriteEvt

Function block for writing of records into a database.

Syntax

Definition:

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

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

## Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>
**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write [176]</td>
<td>Local</td>
<td>Creates a record in the standard table structure specified by Beckhoff.</td>
</tr>
<tr>
<td>WriteBySymbol [177]</td>
<td>Local</td>
<td>Reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff.</td>
</tr>
<tr>
<td>WriteStruct [178]</td>
<td>Local</td>
<td>Creates a record with any table structure.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Write**

This method creates a record in the standard table structure specified by Beckhoff.

**Syntax**

```plaintext
METHOD Write : BOOL
VAR_INPUT
  hDBID: UDINT;
  sTableName: T_MaxString;
  pValue: POINTER TO BYTE;
  cbValue: UDINT;
  sDBSymbolName: T_MaxString;
  eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
  nRingBuffParameter: UDINT;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>pValue</td>
<td>POINTER TO BYTE</td>
<td>Address of the variable to be logged in the standard table structure.</td>
</tr>
<tr>
<td>cbValue</td>
<td>UDINT</td>
<td>Length of the variable to be logged.</td>
</tr>
<tr>
<td>sDBSymbolName</td>
<td>T_MaxString</td>
<td>Name that is logged in the table.</td>
</tr>
<tr>
<td>eDBWriteMode</td>
<td>E_WriteMode</td>
<td>Indicates the write mode. (append, update, ring buffer)</td>
</tr>
<tr>
<td>nRingBuffParameter</td>
<td>UDINT</td>
<td>Additional parameter(s) for the &quot;ring buffer&quot; write mode.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

This sample shows how to use the FB_PLCDBWriteEvt.Write method:

```plaintext
VAR
  fbPLCDBWrite : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
  myValue     : LREAL := 43.23;
  tcMessage   : I_TcMessage;
END_VAR
```
IF fbPLCDBWrite.Write(
    hDBID:= 1,
    sTableName:= 'myTable_WithLReal',
    pValue:= ADR(myValue),
    cbValue:= SIZEOF(myValue),
    sDBSymbolName:= 'MyValue',
    eDBWriteMode:= E_WriteMode.eADS_TO_DB_RingBuff_Count,
    nRingBuffParameter:= 3)
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF

Result in the database:

<table>
<thead>
<tr>
<th>ID</th>
<th>Timestamp</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>Has been dropped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>'2018-01-30 14:04:19'</td>
<td>'MyValue'</td>
<td>41.23</td>
</tr>
<tr>
<td>29</td>
<td>'2018-01-30 14:04:29'</td>
<td>'MyValue'</td>
<td>42.23</td>
</tr>
<tr>
<td>30</td>
<td>'2018-01-30 14:04:39'</td>
<td>'MyValue'</td>
<td>43.23</td>
</tr>
</tbody>
</table>

With the ring buffer option, only three entries of this name are in the database at any one time. Older entries are deleted.

**WriteBySymbol**

This method reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff. ADS symbols from other ADS devices can also be read.

**Syntax**

METHOD WriteBySymbol : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    stADSDevice: ST_ADSDevice;
    stSymbol: ST_Symbol;
    eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
    nRingBuffParameter: UDINT;
END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>stADSDevice</td>
<td>ST_ADSDevice</td>
<td>ADS device from which a symbol is to be logged in the standard table structure.</td>
</tr>
<tr>
<td>stSymbol</td>
<td>ST_Symbol</td>
<td>Symbol name of the variable to be written</td>
</tr>
<tr>
<td>eDBWriteMode</td>
<td>E_WriteMode</td>
<td>Indicates the write mode. (append, update, ring buffer)</td>
</tr>
<tr>
<td>nRingBuffParameter</td>
<td>UDINT</td>
<td>Additional parameter(s) for the &quot;ring buffer&quot; write mode</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteBySymbol</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>
Sample

This sample shows how to use the FB_PLCDBWriteEvt.WriteBySymbol method:

```pascal
VAR
  fbPLCDBWrite : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
  myValue : LREAL := 43.23;
  myAdsDevice : ST_ADSDevice;
  mySymbol : ST_Symbol;
  tcMessage : I_TcMessage;
END_VAR

// Set ADSDevice Information
myAdsDevice.sDevNetID := '127.0.0.1.1.1';
myAdsDevice.nDevPort := 851;
myAdsDevice.eADSRdWrtMode := E_ADSRdWrtMode.bySymbolName;
myAdsDevice.tTimeout := T#5S;

// Set Symbol Information
mySymbol.eDataType := E_PLCDataType.eType_LREAL;
mySymbol.sDBSymbolName := 'MySymbol';
mySymbol.sSymbolName := 'MAIN.myValue';
mySymbol.nBitSize := 8;

// Call Functionblock
IF fbPLCDBWrite.WriteBySymbol(
  hDBID:= 1,
  sTableName:= 'myTable_WithLReal',
  stADSDevice:= myAdsDevice,
  stSymbol:= mySymbol,
  eDBWriteMode:= E_WriteMode.eADS_TO_DB_Append,
  nRingBuffParameter:= 1)
THEN
  IF fbPLCDBWrite.bError THEN
    tcMessage := fbPLCDBWrite.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF

Result in the database:

<table>
<thead>
<tr>
<th>ID</th>
<th>Timestamp</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>'2018-01-30 14:04:19'</td>
<td>'MyValue'</td>
<td>41.23</td>
</tr>
<tr>
<td>29</td>
<td>'2018-01-30 14:04:29'</td>
<td>'MyValue'</td>
<td>42.23</td>
</tr>
<tr>
<td>30</td>
<td>'2018-01-30 14:04:39'</td>
<td>'MyValue'</td>
<td>43.23</td>
</tr>
<tr>
<td>31</td>
<td>'2018-01-30 14:06:12'</td>
<td>'MySymbol'</td>
<td>86.2</td>
</tr>
</tbody>
</table>

WriteStruct

This method creates a record with a freely selectable table structure.

Syntax

METHOD WriteStruct : BOOL
VAR_INPUT
  hDBID: UDINT;
  sTableName: T_MaxString;
  pRecord: POINTER TO BYTE;
  cbRecord: UDINT;
  pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
  cbColumnNames: UDINT;
END_VAR
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>pRecord</td>
<td>POINTER TO BYTE</td>
<td>Address of a structure that is to be logged in a freely selectable table structure.</td>
</tr>
<tr>
<td>cbRecord</td>
<td>UDINT</td>
<td>Length of the structure to be written</td>
</tr>
<tr>
<td>pColumnNames</td>
<td>POINTER TO ARRAY [0..MAX_DBCOLUMNS ] OF STRING(50)</td>
<td>Address of the array containing the column name to be filled.</td>
</tr>
<tr>
<td>cbColumnNames</td>
<td>UDINT</td>
<td>Length of the column name array</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteStruct</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

This sample shows how to use the method FB_PLCDBWriteEvt.WriteStruct:

```pascal
VAR
  fbPLCDBWrite : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
  myRecord     : ST_Record;
  ColumnNames  : ARRAY[0..4] OF STRING(50);
  systime      : GETSYSTEMTIME;
  currentTime  : T_FILETIME;
  tcMessage    : I_TcMessage;
END_VAR

TYPE ST_Record :
  STRUCT
    nID       : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName     : STRING;
    nSensor1  : LREAL;
    nSensor2  : LREAL;
  END_STRUCT
END_TYPE

// set Values
systime(timeLoDw => currentTime.dwLowDateTime, timeHiDW => currentTime.dwHighDateTime);
myRecord.dtTimestamp := FILETIME_TO_DT(currentTime);
myRecord.sName := 'MyStructVal';
myRecord.nSensor1 := 12.34;
myRecord.nSensor2 := 102.5;

// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';

// Call Functionblock
IF fbPLCDBWrite.WriteStruct(
  hDBID:= 1,
  sTableName:= 'myTable_Struct',
  pRecord:= ADR(myRecord),
  cbRecord:= SIZEOF(myRecord),
  pColumnNames:= ADR(ColumnNames) ,
  cbColumnNames:= SIZEOF(ColumnNames))
THEN
  IF fbPLCDBWrite.bError THEN
    tcMessage := fbPLCDBWrite.ipTcResult;
    nState := 255;
  ELSE
```
Result in the database:

<table>
<thead>
<tr>
<th>ID</th>
<th>Timestamp</th>
<th>Name</th>
<th>Sensor1</th>
<th>Sensor2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>'2018-01-30 15:23:26'</td>
<td>'MyStructVal'</td>
<td>12.34</td>
<td>102.5</td>
</tr>
</tbody>
</table>

### 6.1.1.2.6 FB_PLCDBCmdEvt

Function block with two methods. Users can define and transfer their own SQL commands. Placeholders in the SQL command can correlate with structures in the PLC, which reflect the table structure. The database server enters the current data of the structure into the SQL command.

**Syntax**

**Definition:**

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

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Local</td>
<td>Sends any SQL commands to the database. Returned records cannot be read.</td>
</tr>
<tr>
<td>ExecuteDataReturn</td>
<td>Local</td>
<td>Sends any SQL commands to the database. A specified number of records can be read.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Execute

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are replaced by the TwinCAT Database Server with the corresponding values before the execution. Returned records cannot be read.

Syntax

METHOD Execute : BOOL
VAR_INPUT
  hDBID: UDINT;
  pExpression: POINTER TO BYTE;
  cbExpression: UDINT;
  pData: POINTER TO BYTE;
  cbData: UDINT;
  pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
  cbParameter: UDINT;
END_VAR

Sample

VAR
  fbPLCDBCmd : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
  sCmd       : STRING (1000);
  myStruct   : ST_DataAll;
  aPara       : ARRAY[0..14] OF ST_ExpParameter;
  tcMessage  : I_TcMessage;
END_VAR

TYPE ST_DataAll :
  STRUCT
    colBigInt: LINT;
    colInteger: DINT;
    colSmallInt: INT;
    colTinyInt: BYTE;
    colBit: BOOL;
    colMoney: LREAL;
    colFloat: LREAL;
    colReal: REAL;
    colDateTime: DT;
    colNText: STRING(255);
    colNChar: STRING(10);
    colImage: ARRAY[0..255] OF BYTE;
ExecuteDataReturn

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are replaced by the TwinCAT Database Server with the corresponding values before the execution. A specified number of records can be read.

Syntax

METHOD ExecuteDataReturn : BOOL
VAR_INPUT
  hDBID: UDINT;
  pExpression: POINTER TO BYTE;
  cbExpression: UDINT;
  pData: POINTER TO BYTE;

  // set command
  sCmd := 'INSERT INTO MyTableName (colInteger, colSmallInt, colTinyInt, colBit, colMoney, colFloat, colReal, colDateTime, colNText, colNChar, colImage, colNVarChar, colBinary, colVarBinary) VALUES ('
    '('
      {colDateTime}, {colNText}, {colNChar}, {colImage}, {colNVarChar}, {colBinary}, {colVarBinary})'
    ');'
  // call functionblock
  IF fbPLCDBCmd.Execute(
    hDBID:= 1,
    pExpression:= ADR(sCmd),
    cbExpression: SIZEOF(sCmd),
    pData:= ADR(myStruct),
    cbData: SIZEOF(myStruct),
    pParameter:= ADR(aPara),
    cbParameter:= SIZEOF(aPara))
  THEN
    IF fbPLCDBCmd.bError THEN
      tcMessage := fbPLCDBCmd.ipTcResult;
      nState := 255;
    ELSE
      nState := 0;
    END_IF
  END_IF
cbData: UDINT;
pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
cbParameter: UDINT;
nStartIndex: UDINT;
nRecordCount: UDINT;
pReturnData: POINTER TO BYTE;
cbReturnData: UDINT;
pRecords: POINTER TO UDINT;
END_VAR

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>pExpression</td>
<td>POINTER TO BYTE</td>
<td>Address of the string variable with the SQL command</td>
</tr>
<tr>
<td>cbExpression</td>
<td>UDINT</td>
<td>Length of the string variable with the SQL command</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure with the parameter values</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Length of the structure with the parameter values</td>
</tr>
<tr>
<td>pParameter</td>
<td>POINTER TO ARRAY[0..MAX_</td>
<td>Address of the structure array with the parameter values</td>
</tr>
<tr>
<td></td>
<td>DBCOLUMNS] OF ST_ExpParameter</td>
<td></td>
</tr>
<tr>
<td>cbParameter</td>
<td>UDINT</td>
<td>Length of the structure array with the parameter information</td>
</tr>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pReturnData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbReturnData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
<tr>
<td>pRecords</td>
<td>POINTER TO BYTE</td>
<td>Number of read records.</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteDataReturn</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

### Parameterizing the command

The column names for the individual parameters are specified in curly brackets in the SQL command.

Sample: `SELECT * FROM MyHouse_Temperatures WHERE Room = '{SelectedRoom}'`.

Accordingly, SelectedRoom has to be specified as parameter name in the structure ST_ExpParameter.

Some databases do not support the parameterization of SQL clauses. (TOP/LIMIT/ROWNUM/...)

Parameterizable table names are not usually supported.

### Sample

```plaintext
VAR
  fbPLCDBCmd : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
  sCmd      : STRING (1000);
  stPara     : ST_ExpParameter;
  RecordAmt : ULINT := 3;
  ReturnDataStruct : ARRAY [0..9] OF ST_DataAll;
  nRecords  : UDINT;
  tcMessage : I_TcMessage;
END_VAR

// set Parameter configuration
stPara.eParaType := E_ExpParameterType.Int64;
stPara.nParaSize := 8;
stPara.sParaName := 'RecordAmt';
```
PLC API

// set command with placeholder
sCmd := 'SELECT TOP ({RecordAmt}) * FROM MyTableName';

// call functionblock
IF fbPLCDBCmd.ExecuteDataReturn(
    hDBID:= 1,
    pExpression:= ADR(sCmd),
    cbExpression:= SIZEOF(sCmd),
    pData:= ADR(RecordAmt),
    cbData:= SIZEOF(RecordAmt),
    pParameter:= ADR(stPara),
    cbParameter:= SIZEOF(stPara),
    nStartIndex:= 0,
    nRecordCount:= 10,
    pReturnData:= ADR(ReturnDataStruct),
    cbReturnData:= SIZEOF(ReturnDataStruct),
    pRecords:= ADR(nRecords))
THEN
    IF fbPLCDBCmd.bError THEN
        tcMessage := fbPLCDBCmd.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF

6.1.1.3 SQL Expert mode

Fig. 1:
6.1.1.3.1  FB_ConfigTcDBSrvEvt

Function block for creating, reading and deleting configuration entries for the TwinCAT Database Server.

Syntax

Definition:

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

Inputs

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

Outputs

Name | Type | Description
--- | --- | ---
\textit{bBusy} | BOOL | TRUE as soon as a method of the function block is active.
\textit{bError} | BOOL | TRUE when an error occurs.
\textit{ipTcResult} | Tc3_EventLogger.I_TcMessage | Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.

Properties

Name | Type | Access | Description
--- | --- | --- | ---
\textit{eTraceLevel} | TcEventSeverity | Get, Set | Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.

Methods

Name | Definition location | Description
--- | --- | ---
Create [186] | Local | Creates new entries in the XML configuration file for the TwinCAT Database Server
Read [187] | Local | Reads the current configuration of the TwinCAT Database Server
Delete [188] | Local | Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server
Create

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

Syntax

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

Inputs

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

Creating AutoLog groups is currently not supported.

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

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

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

Syntax

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

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDBConfig</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB</td>
<td>Pointer address of the array into which the database configurations are to be written.</td>
</tr>
<tr>
<td>cbDBConfig</td>
<td>UDINT</td>
<td>Length of the database configuration array</td>
</tr>
<tr>
<td>pAutoLogGrpConfig</td>
<td>POINTER TO ARRAY[1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp</td>
<td>Pointer address of the array into which the AutoLogGrp configurations are to be written.</td>
</tr>
<tr>
<td>cbAutoLogGrpConfig</td>
<td>UDINT</td>
<td>Length of the AutoLogGrp configuration array</td>
</tr>
<tr>
<td>pDBCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of database configurations.</td>
</tr>
<tr>
<td>pAutoLogGrpCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of AutoLogGrp configurations.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

IF fbConfigTcDBSrv.Read(
   pDBConfig := ADR(aDBConfig),
   cbDBConfig := SIZEOF(aDBConfig),
   pAutoLogGrpConfig := ADR(aAutoGrpConfig),
   cbAutoLogGrpConfig := SIZEOF(aAutoGrpConfig),
   pDBCount := ADR(nDbCount),
Delete

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

Syntax

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

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTcDBSrvConfigType</td>
<td>E_TcDBSrvConfigType</td>
<td>Type of the configuration to be deleted (database / AutoLog group)</td>
</tr>
<tr>
<td>hConfigID</td>
<td>UDINT</td>
<td>ID of the configuration to be deleted (hDBID or hAutoLogGrpID)</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

6.1.1.3.2 FB_SQLDatabaseEvt

Function block for opening, closing and managing a database connection.
Syntax

Definition:

FUNCTION BLOCK FB_SQLDatabaseEvt
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Local</td>
<td>Opens a connection to a declared database.</td>
</tr>
<tr>
<td>CreateCmd</td>
<td>Local</td>
<td>Initializes an instance of the function block FB_SQLCommandEvt</td>
</tr>
<tr>
<td>CreateSP</td>
<td>Local</td>
<td>Initializes an instance of the function block FB_SQLStoredProcedureEvt.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Local</td>
<td>Closes the connection to the database that was opened by this function block instance.</td>
</tr>
</tbody>
</table>
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Connect

This method opens a connection to a declared database.

Syntax

METHOD Connect : BOOL
VAR_INPUT
   hDBID: UDINT := 1;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

VAR
   fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR

// open connection
IF fbSqlDatabase.Connect(1) THEN
   IF fbSqlDatabase.bError THEN
      nState := 255;
   ELSE
      nState := nState+1;
   END_IF
END_IF

CreateCmd

This method is used to initialize an instance of the function block FB_SQLCommand with the already open database connection of the function block FB_SQLDatabase. The function block FB_SQLCommand only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLCommand can be initialized with the same database connection.

The initialization of the function block FB_SQLCommand is completed in the same cycle. This means that neither the Busy flag of the function block nor the method return value of the CreateCmd method have to be checked.

Syntax

METHOD CreateCmd : BOOL
VAR_INPUT
   pSQLCommand: POINTER TO FB_SQLCommandEvt;
END_VAR
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSQLCommand</td>
<td>POINTER TO FB_SQLCommand</td>
<td>Returns a new instance of the function block FB_SQLCommandEvt.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCmd</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

```plaintext
VAR
  fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR

// create a command reference
IF fbSqlDatabase.CreateCmd(ADR(fbSqlCommand)) THEN
  IF fbSqlDatabase.bError THEN
    nState := 255;
  ELSE
    nState := nState+1;
  END_IF
END_IF
```

FB_SQLCommandEvt [192] can then be used for the execution.

CreateSP

This method is used to initialize an instance of the function block FB_SQLStoredProcedureEvt with the already open database connection of the function block FB_SQLDatabaseEvt. The function block FB_SQLStoredProcedureEvt only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLStoredProcedureEvt can be initialized with the same database connection.

The initialization of the function block FB_SQLStoredProcedureEvt may take several cycles. The Busy flag of the function block or the method return value of the CreateCmd method have to be checked before the function block can be used.

Syntax

```plaintext
METHOD CreateSP : BOOL
VAR_INPUT
  sProcedureName: T_MaxString;
  pParameterInfo: POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter;
  cbParameterInfo: UDINT;
  pSQLProcedure: POINTER TO FB_SQLStoredProcedureEvt;
END_VAR
```

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sProcedureName</td>
<td>T_MaxString</td>
<td>Indicates the name of the procedure to be executed.</td>
</tr>
<tr>
<td>pParameterInfo</td>
<td>POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter</td>
<td>Pointer address for the parameter info list.</td>
</tr>
<tr>
<td>cbParameterInfo</td>
<td>UDINT</td>
<td>Indicates the length of the parameter info list.</td>
</tr>
<tr>
<td>pSQLProcedure</td>
<td>POINTER TO FB_SQLStoredProcedureEvt</td>
<td>Returns a new instance of the function block FB_SQLStoredProcedureEvt.</td>
</tr>
</tbody>
</table>
Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateSP</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

```plaintext
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
    ParaInfo : ST_SQLSPParameter;
END_VAR
ParaInfo.sParameterName := '@Customer_ID';
ParaInfo.eParameterType := E_SPParameterType.Input;
ParaInfo.eParameterDataType := E_ColumnType.BigInt;
ParaInfo.nParameterSize := 8;
IF fbSqlDatabase.CreateSP('dbo.SP_GetCustomerPositions', ADR(ParaInfo), SIZEOF(ParaInfo), ADR(fbSQLStoredProcedure)) THEN
    IF fbSqlDatabase.bError THEN
        nState:=255;
    ELSE
        nState:= nState+1;
    END_IF
END_IF
```

Subsequently, the FB_SQLStoredProcedureEvt can be used to execute the stored procedure.

Disconnect

This method closes the connection to the database that was opened by this function block instance.

Syntax

```plaintext
METHOD Disconnect : BOOL
```

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disconnect</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

```plaintext
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR
// disconnect from database
IF fbSqlDatabase.Disconnect() THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

6.1.1.3.3 FB_SQLCommandEvt
Function block for executing SQL commands. Before it can be used it has to be initialized with the function block FB_SQLDatabaseEvt.

**Syntax**

**Definition:**

```plaintext
FUNCTION BLOCK FB_SQLCommandEvt
VAR_INPUT
   sNetID: T_AmsNetID := '';
   tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
   bBusy: BOOL;
   bError: BOOL;
   ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Local</td>
<td>Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabaseEvt.</td>
</tr>
<tr>
<td>ExecuteDataReturn</td>
<td>Local</td>
<td>Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabaseEvt. An instance of the function block FB_SQLResultEvt can be transferred for reading the returned records.</td>
</tr>
</tbody>
</table>
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Execute

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase.

Syntax

METHOD Execute : BOOL
VAR_INPUT
    pSQLCmd: POINTER TO BYTE;
    cbSQLCmd: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSQLCmd</td>
<td>POINTER TO BYTE</td>
<td>Indicates the pointer address of a string variable with the SQL command to be executed.</td>
</tr>
<tr>
<td>cbSQLCmd</td>
<td>UDINT</td>
<td>Indicates the length of a SQL command to be executed.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>POINTER TO BYTE</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

Uses the command created by FB_SQLDatabaseEvt.CreateCmd() [↓188].

```plaintext
VAR
    fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
    tcMessage : I_TcMessage;
END_VAR

// you can generate this with the SQL Query Editor
sCmd := 'INSERT INTO myTable_Double ( Timestamp, Name, Value) VALUES ( $'2018-01-31 14:59:27$', $'Temperature$', 21.3)';

// call sql command
IF fbSQLCommand.Execute(ADR(sCmd), SIZEOF(sCmd)) THEN
    IF fbSQLCommand.bError THEN
        tcMessage := fbSQLCommand.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

ExecuteDataReturn

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the function block FB_SQLResult can be transferred for reading the returned records.
Syntax

METHOD ExecuteDataReturn : BOOL
VAR_INPUT
  pSQLCmd: POINTER TO BYTE;
  cbSQLCmd: UDINT;
  pSQLDBResult: POINTER TO FB_SQLResult;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSQLCmd</td>
<td>POINTER TO BYTE</td>
<td>Indicates the pointer address of a string variable with the SQL command to be executed.</td>
</tr>
<tr>
<td>cbSQLCmd</td>
<td>UDINT</td>
<td>Indicates the length of a SQL command to be executed.</td>
</tr>
<tr>
<td>pSQLDBResult</td>
<td>POINTER TO FB_SQLResult</td>
<td>Returns a new instance of the function block FB_SQLResult.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteDataReturn</td>
<td>POINTER TO BYTE</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

Uses the command created by FB_SQLDatabaseEvt.CreateCmd().

VAR
  fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
  tcMessage    : I_TcMessage;
END_VAR

// you can generate this with the SQL Query Editor
sCmd := 'SELECT ID, Timestamp, Name, Value FROM myTable_Double';

// call sql command
IF fbSqlCommand.ExecuteDataReturn(ADR(sCmd), SIZEOF(sCmd), ADR(fbSqlResult)) THEN
  IF fbSqlCommand.bError THEN
    nState := 255;
  ELSE
    tcMessage := fbSqlCommand.ipTcResult;
    nState := nState+1;
  END_IF
END_IF

FB_SQLResultEvt can then be used to read the data.

6.1.1.3.4 FB_SQLResultEvt

The function block is used for reading the cached records.

Syntax

Definition:

FUNCTION BLOCK FB_SQLResultEvt
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;

PLC API

END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Local</td>
<td>Reads a specified number of records from the result data cached in the TwinCAT Database Server.</td>
</tr>
<tr>
<td>Release</td>
<td>Local</td>
<td>Releases data buffered by the TwinCAT Database Server.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Read**

This method reads a specified number of records from the result data cached in the TwinCAT Database Server.

**Syntax**

```plaintext
METHOD Read : BOOL
VAR_INPUT
  nStartIndex: UDINT := 0;
  nRecordCount: UDINT := 1;
  pData: POINTER TO BYTE;
```
cbData: UDINT;
bWithVerifying: BOOL := FALSE;
bDataRelease: BOOL := TRUE;
END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
<tr>
<td>bWithVerifying</td>
<td>BOOL</td>
<td>Return data are compared with the pData structure array and adjusted if necessary.</td>
</tr>
<tr>
<td>bDataRelease</td>
<td>BOOL</td>
<td>Releases the cached data.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

```plaintext
VAR
  fbSqlResult : FB_SQLResultEvt(sNetID='', tTimeout := T#5S);
  aReadStruct : ARRAY[1..5] OF ST_StandardRecord;
END_VAR

// get values from internal tc db srv storage
IF fbSqlResult.Read(2, 3, ADR(aReadStruct), SIZEOF(aReadStruct), FALSE, TRUE) THEN
  IF fbSqlResult.bError THEN
    nState := 255;
  ELSE
    nState := nState+1;
  END_IF
END_IF

Result in the PLC:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aReadStruct</td>
<td>ARRAY [1..5] OF ST...</td>
<td></td>
</tr>
<tr>
<td>aReadStruct[1]</td>
<td>ST_StandardRecord</td>
<td></td>
</tr>
<tr>
<td>nID</td>
<td>LINT</td>
<td>9</td>
</tr>
<tr>
<td>dTimestamp</td>
<td>DATE_AND_TIME</td>
<td>DT#2018-1-31 15:4:59</td>
</tr>
<tr>
<td>sName</td>
<td>STRING(80)</td>
<td>'Temperature'</td>
</tr>
<tr>
<td>rValue</td>
<td>LREAL</td>
<td>21.3</td>
</tr>
<tr>
<td>aReadStruct[2]</td>
<td>ST_StandardRecord</td>
<td></td>
</tr>
<tr>
<td>nID</td>
<td>LINT</td>
<td>10</td>
</tr>
<tr>
<td>dTimestamp</td>
<td>DATE_AND_TIME</td>
<td>DT#2018-1-31 15:5:59</td>
</tr>
<tr>
<td>sName</td>
<td>STRING(80)</td>
<td>'Temperature'</td>
</tr>
<tr>
<td>rValue</td>
<td>LREAL</td>
<td>21.2</td>
</tr>
</tbody>
</table>

**Release**

This method can be used to release data cached by the TwinCAT Database Server.
Syntax

METHOD Release : BOOL

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

6.1.1.3.5 FB_SQLStoredProcedureEvt

Function block for executing stored procedures of the database. Before it can be used it has to be initialized with the function block FB_SQLDatabaseEvt.

Syntax

Definition:

```plaintext
FUNCTION BLOCK FB_SQLStoredProcedureEvt
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>


Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSev</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
<tr>
<td></td>
<td>erity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>222</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Local</td>
<td>Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabaseEvt.</td>
</tr>
<tr>
<td>ExecuteDataReturn</td>
<td>Local</td>
<td>Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabaseEvt. An instance of the function block FB_SQLResultEvt can be transferred for reading the returned records.</td>
</tr>
<tr>
<td>Release</td>
<td>Local</td>
<td>Releases the parameter information of the stored procedure that was transferred during initialization.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Execute

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabaseEvt.

Syntax

```plaintext
METHOD Execute : BOOL
VAR_INPUT
   pParameterStrc: POINTER TO BYTE;
   cbParameterStrc: UDINT;
END_VAR
```

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pParameterStrc</td>
<td>POINTER TO BYTE</td>
<td>Pointer address to the parameter structure that is transferred to the procedure.</td>
</tr>
<tr>
<td>cbParameterStrc</td>
<td>UDINT</td>
<td>Length of the parameter structure.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>
Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [188].

```plaintext
VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID=' ', tTimeout := T#5S);
    Customer_ID : LINT;
    tcMessage : I_TcMessage;
END_VAR

IF fbSQLStoredProcedure.Execute(pParameterStrc := ADR(Customer_ID) , cbParameterStrc := SIZEOF(Customer_ID)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

ExecuteDataReturn

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the FB_SQLResult function block can be transferred for reading the returned records.

Syntax

```plaintext
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
    pSQLDBResult: POINTER TO FB_SQLDBResultEvt;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pParameterStrc</td>
<td>POINTER TO BYTE</td>
<td>Pointer address to the parameter structure that is transferred to the procedure.</td>
</tr>
<tr>
<td>cbParameterStrc</td>
<td>UDINT</td>
<td>Length of the parameter structure</td>
</tr>
<tr>
<td>pSQLDBResult</td>
<td>POINTER TO FB_SQLDBResultEvt</td>
<td>Returns a new instance of the function block FB_SQLDBResultEvt.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [188].

```plaintext
VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID=' ', tTimeout := T#5S);
    Customer_ID : LINT;
    tcMessage : I_TcMessage;
END_VAR

IF fbSQLStoredProcedure.ExecuteDataReturn(pParameterStrc := ADR(Customer_ID) , cbParameterStrc := SIZEOF(Customer_ID), pSQLDBResult := ADR(fbSqlResult)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
```
nState := nState+1;
END_IF
END_IF

FB_SQLResultEvt [195] can then be used to read the data.

Release

This method releases the parameter information of the stored procedure, which was transferred during initialization.

Syntax

METHOD Release : BOOL

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

6.1.1.4 NoSQL Expert Mode
6.1.4.1 Query Builder

In order to support as many NoSQL databases as possible, there are query function blocks that offer different parameterizations for queries. These function blocks are then passed as an interface to the methods of the FB_NoSQLQuery.

**FB_NoSQLQueryBuilder_DocumentDB**

Function block for defining a query for the database. The query is sent with FB_NoSQLQueryEvt [204]. It is not necessary to call the Build method.

### Syntax

**Definition:**

```
FUNCTION_BLOCK FB_NoSQLQueryBuilder_DocumentDB
VAR_INPUT
  eQueryType : E_DocumentDbQueryType;
  sCollectionName : T_MAXSTRING;
  pQueryOptions: POINTER TO BYTE;
  cbQueryOptions : UDINT;
END_VAR
VAR_OUTPUT
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eQueryType</td>
<td>E_DocumentDbQueryType</td>
<td>Type of query sent to the database.</td>
</tr>
<tr>
<td>sCollectionName</td>
<td>T_MAXSTRING</td>
<td>Name of the collection that is the target of the query.</td>
</tr>
<tr>
<td>pQueryOptions</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address for the query options [231].</td>
</tr>
<tr>
<td>cbQueryOptions</td>
<td>UDINT</td>
<td>Length of the query options.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build [203]</td>
<td>Local [optional]</td>
<td>This method generates a query for the function block FB_NoSQLQueryEvt [204] from the set parameters.</td>
</tr>
</tbody>
</table>

**Sample:**

```
VAR
  fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
  sFilter : T_MAXSTRING;
  stOptions : T_QueryOptionDocumentDB_Find;
END_VAR
```
// Set your settings before you run the query
stOptions.pFilter := ADR(sFilter);
stOptions.cbFilter := SIZEOF(sFilter);

fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.Find;
fbNoSQLQueryBuilder_DocumentDB.sCollectionName := 'MyCollectionName';
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions := ADR(stOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(stOptions);

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Build

This method is called automatically in case of a FB_NoSQLQueryEvt (either with Execute or ExecuteDataReturn) before the query is sent. It creates a TwinCAT 3 Database Server-specific query from the specified parameters of the QueryBuilder.

FB_NoSQLQueryBuilder_TimeSeriesDB

Function block for defining a query for a TimeSeries database. The query is sent with FB_NoSQLQueryEvt. It is not necessary to call the Build method. Data structures can be described with attributes to affect individual settings.

_inputs_

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pQueryOptions</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address for the query options.</td>
</tr>
<tr>
<td>cbQueryOptions</td>
<td>UDINT</td>
<td>Length of the query options.</td>
</tr>
</tbody>
</table>

Syntax

Definition:

FUNCTION BLOCK FB_NoSQLQueryBuilder_TimeSeriesDB
VAR_INPUT
   pQueryOptions : POINTER TO BYTE;
   cbQueryOptions : UDINT;
END_VAR
VAR_OUTPUT
END_VAR

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build</td>
<td>Local</td>
<td>[optional] This method generates a query for the function block FB_NoSQLQueryEvt from the set parameters.</td>
</tr>
</tbody>
</table>
Example:

```plaintext
VAR
  fbNoSQLQueryBuilder_TimeSeriesDB : FB_NoSQLQueryBuilder_TimeSeriesDB;
  QueryOption_TSDB_Insert : T_QueryOptionTimeSeriesDB_Insert;
  fbNoSqlQueryEvt : FB_NoSQLQueryEvt(sNetID := '', tTimeout := T#15S);
  MyStructArray: ARRAY[1..1000] OF MyStruct;
END_VAR

CASE nState OF
  1: // init
    fbNoSQLQueryBuilder_TimeSeriesDB.pQueryOptions := ADR(QueryOption_TSDB_Insert);
    fbNoSQLQueryBuilder_TimeSeriesDB.cbQueryOptions := SIZEOF(QueryOption_TSDB_Insert);
    QueryOption_TSDB_Insert.sTableName := 'MeasurementName';
    QueryOption_TSDB_Insert.sDataType := 'MyStruct';
    QueryOption_TSDB_Insert.pSymbol := ADR(MyStructArray);
    QueryOption_TSDB_Insert.cbSymbol := SIZEOF(MyStructArray);
    QueryOption_TSDB_Insert.nDataCount := 1000; // ArrayLength
    QueryOption_TSDB_Insert.nStartTimestamp := F_GetSystemTime(); // get current twincat time
    QueryOption_TSDB_Insert.nCycleTime := 1000; // equivalent to 1 ms
  2: // write values
    IF fbNoSqlQueryEvt.Execute(dbid, fbNoSQLQueryBuilder_TimeSeriesDB) THEN
      IF fbNoSqlQueryEvt.bError THEN
        // do some error handling here
      ELSE
        // success
      END_IF
    END_IF
END_CASE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Build**

This method is called automatically in case of a FB_NoSQLQueryEvt (either with Execute or ExecuteDataReturn) before the query is sent. It creates a TwinCAT 3 Database Server-specific query from the specified parameters of the QueryBuilder.

### 6.1.1.4.2 FB_NoSQLQueryEvt

Function block for executing NoSQL database queries. A QueryBuilder function block that describes the query is used as the input parameter for the methods.

**Syntax**

**Definition:**

```
FUNCTION BLOCK FB_NoSQLQueryEvt
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute [205]</td>
<td>Local</td>
<td>Sends the query created by the QueryBuilder function block to the database.</td>
</tr>
<tr>
<td>ExecuteDataReturn [206]</td>
<td>Local</td>
<td>Sends the query created by the QueryBuilder function block to the database. An instance of the FB_NoSqlResultEvt [207] function block can be transferred for reading the returned records.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Execute**

This method sends a query to the NoSQL database, which was previously set with the I_NoSQLQueryBuilder [202] function block.

**Syntax**

```plaintext
METHOD Execute : BOOL
VAR_INPUT
  hDBID: UDINT;
  iNoSQLQueryBuilder: I_NoSQLQueryBuilder;
END_VAR
```
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>ID of the set database configuration</td>
</tr>
<tr>
<td>iNoSSQLQueryBuilder</td>
<td>I_NoSQLQueryBuilder</td>
<td>Pre-parameterizedQueryBuilder function block. This varies depending on the database.</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Uses the QueryBuilder to execute the corresponding query.

```plaintext
VAR
    fbNoSQLQuery : FB_NoSQLQueryEvt(sNetID := '', tTimeout := T#5S);
    fbNoSQLQueryBuilder_DocumentDB : FB_NoSQLQueryBuilder_DocumentDB;
    myDBID : UDINT := 1;
    sDocument : STRING(1000);
    TcMessage : I_TcMessage;
END_VAR

// set QueryInputs
fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.InsertOne;
fbNoSQLQueryBuilder_DocumentDB.pQueryOptions := ADR(InsertQueryOptions);
fbNoSQLQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(InsertQueryOptions);

// set insert parameter:
sDocument := '{Name : "MyValue", Value : 123.456}';
InsertQueryOptions.pDocuments:= ADR(sDocument);
InsertQueryOptions.cbDocuments:= SIZEOF(sDocument);

// call nosql command
IF fbNoSQLQuery.Execute(myDBID, fbNoSQLQueryBuilder_DocumentDB) THEN
    IF fbNoSQLQuery.bError THEN
        TcMessage := fbNoSQLQuery.ipTcResult
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF
```

First, the FB_NoSQLQueryEvt function block is parameterized via the FB_NoSQLQueryBuilder_DocumentDB function block. Depending on the query type there are various options, such as T_QueryOptionDocumentDB_Insert, for setting the document to be inserted.

### ExecuteDataReturn

This method executes a query to a NoSQL database that was previously set using the I_NoSQLQueryBuilder function block. The transferred instance of type FB_NoSQLResultEvt is filled with return values.

#### Syntax

```plaintext
METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    hDBID : UDINT;
    iNoSSQLQueryBuilder : I_NoSQLQueryBuilder;
    pNoSQLResult : POINTER TO FB_NoSQLResultEvt;
END_VAR
```
### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>ID of the set database configuration</td>
</tr>
<tr>
<td>iNoSQLQueryBuilder</td>
<td>I_NoSQLQueryBuilder</td>
<td>Preconfigured QueryBuilder function block that defines the query to be sent.</td>
</tr>
<tr>
<td>pNoSQLResult</td>
<td>POINTER TO FB_NoSQLResultEvt</td>
<td>Specifies the address for the FB_NoSQLResultEvt function block, which can be used to read the results.</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteDataReturn</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
<tr>
<td>nDataCount</td>
<td>UDINT</td>
<td>[optional] Number of records returned</td>
</tr>
</tbody>
</table>

Uses the QueryBuilder to execute the corresponding query.

```plaintext
VAR
   fbNoSqlQuery : FB_NoSSQLQueryEvt(sNetID := '', tTimeout := T#5S);
   fbNoSqlQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB
   FindQueryOptions : T_QueryOptionDocumentDB_Find;
   fbNoSqlResult : FB_NoSQLResultEvt(sNetID := '', tTimeout := T#5S);
   myDBID : UDINT := 1;
   sFilter : STRING(255);
   sSort : STRING(255);
   sProjection: STRING(255);
   TcMessage : I_TcMessage;
END_VAR

// set QueryInputs:
fbNoSqlQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.Find;
fbNoSqlQueryBuilder_DocumentDB.pQueryOptions := ADR(FindQueryOptions);
fbNoSqlQueryBuilder_DocumentDB.cbQueryOptions := SIZEOF(FindQueryOptions);

//set Find Parameter ([optional] sort, projection):
sFilter := '{}'; // read all data from database
FindQueryOptions.pFilter:= ADR(sFilter);
FindQueryOptions.cbFilter:= SIZEOF(sFilter);

// call nosql query:
IF fbNoSqlQuery.ExecuteDataReturn(myDBID, fbNoSqlQuery, ADR(fbNoSqlResult)) THEN
  IF fbNoSqlQuery.bError THEN
    TcMessage := fbNoSqlQuery.ipTcResult;
    nState := 255;
  ELSE
    nState := nState+1;
  END_IF
END_IF
```

First, the FB_NoSQLQueryEvt function block is parameterized via the FB_NoSQLQueryBuilder_DocumentDB function block. Depending on the query type there are various options, such as T_QueryOptionDocumentDB_Find, for defining the filter, sorting or projection.

### 6.1.1.4.3 FB_NoSQLResultEvt

Function block for reading buffered records.
The records must first be retrieved from the database using the function block `FB_NoSQLQueryEvt` when the `ExecuteDataReturn` method is called. The function block `FB_NoSQLResultEvt` is specified for initialization. They can then be read out either as a PLC structure or as a string.

**Syntax**

**Definition:**

```plaintext
FUNCTION BLOCK FB_SQLResultEvt
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResult: Tc3_EventLogger.I_TcMessage
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
<tr>
<td>nDataCount</td>
<td>UDINT</td>
<td>Indicates the number of returned records available from the call of the function block <code>FB_NoSQLQueryEvt_ExecuteDataReturn()</code></td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadAsString</td>
<td>Local</td>
<td>Reads a specified number of records from the result data cached in the TwinCAT Database Server as JSON string.</td>
</tr>
<tr>
<td>ReadAsStruct</td>
<td>Local</td>
<td>Reads a specified number of records from the result data cached in the TwinCAT Database Server into the specified structure.</td>
</tr>
<tr>
<td>Release</td>
<td>Local</td>
<td>Releases data buffered by the TwinCAT Database Server.</td>
</tr>
</tbody>
</table>
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

ReadAsString

This method reads a specified number of records from the result data cached in the TwinCAT Database Server. An array of strings is specified into which this data is to be copied as JSON.

Syntax

METHOD ReadAsString : BOOL
VAR_INPUT
  nStartIndex: UDINT := 0;
  nRecordCount: UDINT := 1;
  pData: POINTER TO BYTE;
  cbData: UDINT;
  nMaxDocumentSize : UDINT;
  bDataRelease: BOOL := TRUE;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Indicates the address of the string array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the string array in bytes.</td>
</tr>
<tr>
<td>nMaxDocumentSize</td>
<td>UDINT</td>
<td>Indicates the maximum size of a single JSON document from pData.</td>
</tr>
<tr>
<td>bDataRelease</td>
<td>BOOL</td>
<td>Releases the cached data.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadAsString</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample:

VAR
  fbNoSqlResult : FB_NoSQLResultEvt(sNetID := '', tTimeout := T#5S);
  aRead_Json : ARRAY[0..2] OF STRING(1000);
  TcMessage : I_TcMessage;
END_VAR

IF fbNoSqlResult.ReadAsString(
  nStartIndex:= 0,
  nRecordCount:= 3,
  pData:= ADR(aRead_Json),
  cbData:= SIZEOF(aRead_Json),
  nMaxDocumentSize : SIZEOF(aRead_Json[0]),
  bDataRelease:= TRUE)
THEN
  IF fbNoSqlResult.bError THEN
    TcMessage := fbNoSqlResult.ipTcResult;
    nstate := 255;
  ELSE
    nstate := nstate+1;
  END_IF
END_IF
ReadAsStruct

This method reads a specified number of records from the buffered result data. A structure or an array of a structure is specified in which the data is to be written. The data type schema of this structure should correspond as closely as possible to that of the read data. The variable names are compared with those of the record. A validation makes it possible to detect deviations and respond to them.

If there is a requirement to use different names in the database and in the PLC, the names can be described in the structure with the attribute 'ElementName' with the assigned name from the database.

Syntax

METHOD ReadAsStruct: BOOL
VAR_INPUT
  nStartIndex: UDINT := 0;
  nRecordCount: UDINT := 1;
  pData: POINTER TO BYTE;
  cbData: UDINT;
  bValidate: BOOL := FALSE;
  pNoSQLValidation : POINTER TO FB_NoSQLValidationEvt;
  bDataRelease: BOOL := TRUE;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
<tr>
<td>bValidate</td>
<td>BOOL</td>
<td>Return data are compared with the pData structure array and adjusted if necessary.</td>
</tr>
<tr>
<td>pNoSQLValidation</td>
<td>POINTER TO FB_NoSQLValidationEvt</td>
<td>Address of the function block FB_NoSQLValidationEvt that provides further information for validating the call.</td>
</tr>
<tr>
<td>bDataRelease</td>
<td>BOOL</td>
<td>Releases the cached data.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadAsStruct</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample:

```plaintext
VAR
  fbNoSQLResult: FB_NoSQLResultEvt(sNetID := '', tTimeout := T#5S);
  aRead : ARRAY[0..2] OF ST_MyDataStruct;
  fbNoSQLValidation : FB_NoSQLValidationEvt(sNetID := '', tTimeout := #5S);
END_VAR

IF fbNoSQLResult.ReadAsStruct(
  nStartIndex:= 0,
  nRecordCount:= 3,
  pData:= ADR(aRead),
  cbData:= SIZEOF(aRead),
  bValidate:= TRUE,
  pNoSQLValidation:= ADR(fbNoSQLValidation),
  bDataRelease:= TRUE)
THEN
  IF fbNoSQLResult.bError THEN
    TcMessage := fbNoSQLResult.ipTcResult;
    nstate := 255;
  ELSE
    ...;
  END_IF
ELSE
  ...;
END_IF
```
Release

This method can be used to release data cached by the TwinCAT Database Server.

Syntax

METHOD Release : BOOL

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

6.1.1.4.4 FB_NoSQLValidationEvt

Function block for reading the validation events and results that occurred when reading the data with FB_NoSQLResultEvt [207]. This function block is initialized via the CreateValidation method of the NoSQLResult. It refers to the last call of the ReadAsStruct [210] method.

Syntax

Definition:

FUNCTION BLOCK FB_NoSQLValidationEvt
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResult: Tc3_EventLogger.I_TcResultEvent
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>
**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResult</td>
<td>Tc3_EventLogger.I_TcMessage[221]</td>
<td>Message interface from the TwinCAT 3 EventLogger, which provides details on the return value.</td>
</tr>
</tbody>
</table>

**Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity[222]</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetIssues[212]</td>
<td>Local</td>
<td>Reads a list of validation events as a string array.</td>
</tr>
<tr>
<td>GetRemainingData[213]</td>
<td>Local</td>
<td>Reads the data as a string which could not be assigned to any element in the structure in the PLC.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**GetIssues**

Reads the validation events that occurred into an array of type T_MAXSTRING. These contain information about remaining or unassigned records or which elements of the PLC structure were not filled.

**Syntax**

```plaintext
METHOD GetIssues : BOOL
VAR_INPUT
    pData : POINTER TO BYTE;
    cbData: UDINT;
    bDataRelease : BOOL;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the array of type T_MAXSTRING in which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the string array in bytes.</td>
</tr>
<tr>
<td>bDataRelease</td>
<td>BOOL</td>
<td>Releases the cached data.</td>
</tr>
</tbody>
</table>
### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadAsString</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

#### Sample:

```pascal
VAR
    fbNoSqlValidation : FB_NoSQLValidation(sNetID := '', tTimeout := t#15S);
    aIssues : ARRAY[0..1000] OF T_MAXSTRING;
END_VAR

IF fbNoSqlValidation.GetIssues(
    pData:= ADR(aIssues),
    cbData:= SIZEOF(aIssues),
    bDataRelease:= TRUE)
THEN
    IF fbNoSqlValidation.bError THEN
        TcMessage := fbNoSqlValidation.ipTcResult;
        nstate := 255;
    ELSE
        nstate := nstate+1;
    END_IF
END_IF
```

### GetRemainingData

This method can be used to read the remaining data in JSON format after the validation. This includes records which could not be assigned to the PLC structure, for example.

#### Syntax

```pascal
METHOD GetRemainingData : BOOL
VAR_INPUT
    pData : POINTER TO BYTE;
    cbData : UDINT;
    cbDocument : UDINT;
    bDataRelease : BOOL;
END_VAR
```

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Indicates the address of the string array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the string array in bytes.</td>
</tr>
<tr>
<td>cbDocument</td>
<td>UDINT</td>
<td>Specifies the length of the string in the array.</td>
</tr>
<tr>
<td>bDataRelease</td>
<td>BOOL</td>
<td>Releases the cached data.</td>
</tr>
</tbody>
</table>

#### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetRemainingData</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

#### Sample:

```pascal
VAR CONSTANT
    cDocumentSize : UDINT := 1000;
END_VAR
VAR
    fbNoSqlValidation : FB_NoSQLValidation(sNetID := '', tTimeout := t#15S);
    aRemainingData : ARRAY[0..1000] OF STRING(cDocumentSize);
END_VAR
```
If `fbNoSqlValidation.GetRemainingData`
  pData:= ADR(aRemainingData),
  cbData:= SIZEOF(aRemainingData),
  cbDocument:= cDocumentSize,
  bDataRelease:= TRUE)
THEN
  IF fbNoSqlValidation.bError THEN
    TcMessage := fbNoSqlValidation.ipTcResult;
    nstate := 255;
  ELSE
    nstate := nstate+1;
  END_IF
END_IF

**Release**

Releases the validation results in the memory.

**Syntax**

METHOD Release : BOOL

#### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**6.1.1.4.5 Helper**

These function blocks offer useful functions for dealing with data types, for example.

**FB_NoSQLObjectId_MongoDB**

The function block for parsing the ObjectId from the MongoDB. In the PLC it is described by the data type `T_ObjectId_MongoDB` [p. 230].

**Syntax**

FUNCTION_BLOCK FB_NoSQLObjectId_MongoDB
VAR_INPUT
  ObjectId : T_ObjectId_MongoDB;
END_VAR

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectId</td>
<td>T_ObjectId_MongoDB</td>
<td>12-byte data type for describing the ObjectId.</td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
<tr>
<td>nId</td>
<td>UDINT</td>
<td>Non-unique, sequential number</td>
</tr>
<tr>
<td>nMachineId</td>
<td>UDINT</td>
<td>Identification of the machine</td>
</tr>
<tr>
<td>nProcessId</td>
<td>UINT</td>
<td>Identification of the writing process</td>
</tr>
<tr>
<td>tTimestamp</td>
<td>DATE_AND_TIME</td>
<td>Time stamp of the record</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToString</td>
<td>Local</td>
<td>STRING(36)</td>
<td>Returns the ID as a string with type designation. Example: 'ObjectId(&quot;5be15c11afa6ec72b107dafaf&quot;)'</td>
</tr>
<tr>
<td>ValueOf</td>
<td>Local</td>
<td>STRING(24)</td>
<td>Returns only the ID as a string. Example: '5be15c11afa6ec72b107dafaf'</td>
</tr>
</tbody>
</table>

Requirements

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

6.1.1.5 Support of the Tc3_Eventlogger

The TwinCAT 3 Database Server supports the TwinCAT 3 EventLogger (TwinCAT 3 Version 4022.20). This makes it possible to read out details of function block events via an interface. Further information on the EventLogger can be found in the TwinCAT 3 basic libraries.

All function blocks of the TwinCAT 3 Database Server support the interface of the Tc3 EventLogger. The interface Tc3_Eventlogger.I_TcMessage is used as the return value of the function blocks. In addition to the return value, the eTraceLevel property is available to determine the event weighting.

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTraceLevel</td>
<td>TcEventSeverity</td>
<td>Get, Set</td>
<td>Specifies the weighting of the events. Only events with a weighting higher than this value are sent to the TwinCAT system.</td>
</tr>
</tbody>
</table>

Sample:

As an example, the following weighting is specified for the function block FB_PLCDBWrite:

```
fbPLCDBWriteEvt.eTraceLevel := TcEventSeverity.Warning;
```

All events that represent at least a warning are now sent here. Events of the "Information" weighting are ignored in this case.

The Tc3_Database function blocks themselves have the output ipTcResult of data type Tc3_Eventlogger.I_TcMessage. All functions of this interface that are offered can be used.

In this sample, the function block is called first.
1: // Call Functionblock
IF fbPLCDBWriteEvt.WriteStruct(
  hDBID:= 1,
  sTableName:= 'myTable_Struct',
  pRecord:= ADR(myRecord),
  cbRecord:= SIZEOF(myRecord),
  pColumnNames:= ADR(ColumnNames),
  cbColumnNames:= SIZEOF(ColumnNames))
THEN
  IF fbPLCDBWriteEvt.bError THEN
    myTcMessage := fbPLCDBWriteEvt.ipTcResult
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF

If an error occurs, we now want to request the event text in the runtime environment. The RequestEventText method can be used for this purpose. Use \textit{nLangId} = 1031 to read the error code in German. This is one of the many functions of the \textit{Tc3_Eventlogger.I_TcMessage} \[221\] interface.

255://Request EventText
IF myTcMessage.RequestEventText(1031,
  ADR(MyEventString),
  SIZEOF(MyEventString))THEN
  nState := 0;
END_IF

6.1.1.5.1 I_TcEventBase

Methods and properties of an event are defined in this basic interface.

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EqualsTo [[217]]</td>
<td>Compares the event with another instance.</td>
</tr>
<tr>
<td>EqualsToEventClass [[217]]</td>
<td>Compares the event class of the event with another event class.</td>
</tr>
<tr>
<td>EqualsToEventEntryEx [[218]]</td>
<td>Compares the event definition of the event with another event definition.</td>
</tr>
<tr>
<td>GetJsonAttribute [[219]]</td>
<td>Returns the Json attribute.</td>
</tr>
<tr>
<td>RequestEventClassName [[219]]</td>
<td>Requests the name of the event class.</td>
</tr>
<tr>
<td>RequestEventText [[220]]</td>
<td>Returns the text for the event.</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSeverity</td>
<td>TcEventSeverity [[222]]</td>
<td>Get</td>
<td>Returns the severity.</td>
</tr>
<tr>
<td>EventClass</td>
<td>GUID</td>
<td>Get</td>
<td>Returns the GUID of the event class.</td>
</tr>
<tr>
<td>ipSourceInfo</td>
<td>I_TcSourceInfo</td>
<td>Get</td>
<td>Returns a pointer to the source definition.</td>
</tr>
<tr>
<td>nEventId</td>
<td>UDINT</td>
<td>Get</td>
<td>Returns the ID of the event.</td>
</tr>
<tr>
<td>stEventEntry</td>
<td>TcEventEntry</td>
<td>Get</td>
<td>Returns the event definition.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4022.20</td>
<td>PC or CX (x64, x86, ARM)</td>
<td>Tc3_EventLogger</td>
</tr>
</tbody>
</table>
EqualsTo

This method carries out a comparison with another event specified at the input.

Syntax

METHOD EqualsTo : BOOL
VAR_INPUT
  ipOther : I_TcEventBase;
END_VAR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipOther</td>
<td>I_TcEventBase</td>
<td>Event to be compared</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EqualsTo</td>
<td>BOOL</td>
<td>Returns TRUE if the events match.</td>
</tr>
</tbody>
</table>

EqualsToEventClass

This method carries out a comparison with another event class specified at the input.

Syntax

METHOD EqualsToEventClass : BOOL
VAR_INPUT
  OtherEventClass : GUID
END_VAR

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OtherEventClass</td>
<td>GUID</td>
<td>Event class to be compared.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EqualsToEventClass</td>
<td>BOOL</td>
<td>Returns TRUE if the event classes match.</td>
</tr>
</tbody>
</table>
EqualsToEventEntry

This method carries out a comparison with another event specified at the input.

Syntax

METHOD EqualsToEventEntry : BOOL
VAR_INPUT
  OtherEventClass : GUID;
  nOtherEventID : UDINT;
  eOtherSeverity : TcEventSeverity;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OtherEventClass</td>
<td>GUID</td>
<td>Event class of the event to be compared.</td>
</tr>
<tr>
<td>nOtherEventID</td>
<td>UDINT</td>
<td>Event ID of the event to be compared.</td>
</tr>
<tr>
<td>eOtherSeverity</td>
<td>TcEventSeverity</td>
<td>Event severity of the event to be compared.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EqualsToEventEntry</td>
<td>BOOL</td>
<td>Returns TRUE if the events match.</td>
</tr>
</tbody>
</table>

EqualsToEventEntryEx

This method carries out a comparison with another event specified at the input.

Syntax

METHOD EqualsToEventEntryEx : BOOL
VAR_INPUT
  stOther : TcEventEntry;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stOther</td>
<td>TcEventEntry</td>
<td>Event to be compared.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EqualsToEventEntryEx</td>
<td>BOOL</td>
<td>Returns TRUE if the events match.</td>
</tr>
</tbody>
</table>
GetJsonAttribute

This method returns the Json attribute.

Syntax

METHOD GetJsonAttribute : HRESULT
VAR_INPUT
  sJsonAttribute : REFERENCE TO STRING;
  nJsonAttribute : UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sJsonAttribute</td>
<td>REFERENCE TO STRING</td>
<td>Reference to a variable of the type String</td>
</tr>
<tr>
<td>nJsonAttribute</td>
<td>UDINT</td>
<td>Length of the String variable</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetJsonAttribute</td>
<td>HRESULT</td>
<td>Returns S_OK if the method call was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns ERROR_BAD_LENGTH if the length of the variable is too small.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Otherwise HRESULT is returned as error code.</td>
</tr>
</tbody>
</table>

RequestEventClassName

This method returns the name of the event class.

Syntax

METHOD RequestEventClassName : BOOL
VAR_INPUT
  nLangId      : DINT;
  sResult      : REFERENCE TO STRING;
  nResultSize  : UDINT;
END_VAR
VAR_OUTPUT
  bError       : BOOL;
  hrErrorCode  : HRESULT;
END_VAR
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLangId</td>
<td>DINT</td>
<td>Specifies the language ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>English (en-US) = 1033</td>
</tr>
<tr>
<td></td>
<td></td>
<td>German (de-DE) = 1031</td>
</tr>
<tr>
<td>sResult</td>
<td>REFERENCE TO STRING</td>
<td>Reference to a variable of the type String</td>
</tr>
<tr>
<td>nResultSize</td>
<td>UDINT</td>
<td>Size of the String variable in bytes</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestEventClassName</td>
<td>BOOL</td>
<td>Returns TRUE as soon as the request has been terminated. Returns FALSE if the asynchronous request is still active. The method must be called until the return value is TRUE.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>Returns FALSE if the method call was successful. Returns TRUE if an error has occurred.</td>
</tr>
<tr>
<td>hrErrorCode</td>
<td>HRESULT</td>
<td>Returns S_OK if the method call was successful. An error code is output in case of an error.</td>
</tr>
</tbody>
</table>

RequestEventText

This method returns the event text.

Syntax

METHOD RequestEventText : BOOL
VAR_INPUT
    nLangId : DINT;
    sResult : REFERENCE TO STRING;
    nResultSize : UDINT;
END_VAR
VAR_OUTPUT
    bError : BOOL;
    hrErrorCode : HRESULT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLangId</td>
<td>DINT</td>
<td>Specifies the language ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>English (en-US) = 1033</td>
</tr>
<tr>
<td></td>
<td></td>
<td>German (de-DE) = 1031</td>
</tr>
<tr>
<td>sResult</td>
<td>REFERENCE TO STRING</td>
<td>Reference to a variable of the type String</td>
</tr>
<tr>
<td>nResultSize</td>
<td>UDINT</td>
<td>Size of the String variable in bytes</td>
</tr>
</tbody>
</table>
### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestEventText</td>
<td>BOOL</td>
<td>Returns TRUE as soon as the request has been terminated. Returns FALSE if the asynchronous request is still active. The method must be called until the return value is TRUE.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>Returns FALSE if the method call was successful. Returns TRUE if an error has occurred.</td>
</tr>
<tr>
<td>hrErrorCode</td>
<td>HRESULT</td>
<td>Returns S_OK if the method call was successful. An error code is output in case of an error.</td>
</tr>
</tbody>
</table>

### 6.1.1.5.2 I_TcMessage

This interface provides methods and properties for the message handling.

#### Inheritance hierarchy

I_TcEventBase [216]

I_TcMessage

#### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>Sends a message</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1.4022.20</td>
<td>PC or CX (x64, x86, ARM)</td>
<td>Tc3_EventLogger</td>
</tr>
</tbody>
</table>

### Send

This method sends the message.

#### Syntax

METHOD Send : HRESULT
VAR_INPUT
  nTimeStamp: ULIMIT;
END_VAR
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| nTimeStamp | ULINT  | 0: Current time stamp is used<br>
> 0: External time stamp in 100 nanoseconds since January 1st, 1601 (UTC). |

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>FB_HRESULT</td>
<td>Returns S_OK if the method call was successful, otherwise HRESULT as error code</td>
</tr>
</tbody>
</table>

6.1.1.5.3 Data types

TcEventSeverity

Defines the severity of the event.

Syntax

Definition:

```plaintext
{attribute 'qualified_only'}
TYPE TcEventSeverity : {
    Verbose  := 0,
    Info     := 1,
    Warning  := 2,
    Error    := 3,
    Critical := 4);
END_TYPE
```

6.1.2 Data types

6.1.2.1 Config

6.1.2.1.1 E_DatabaseType

Syntax

Definition:

```plaintext
{attribute 'qualified_only'}
TYPE E_DatabaseType : {
    MS_Compact_SQL := 0,
    MS_Access     := 1,
    MS_SQL         := 2,
    ASCII          := 3,
    ODBC_MySQL     := 4,
    ODBC_PostgreSQL := 5,
    ODBC_Oracle    := 6,
    ODBC_DB2       := 7,
    ODBC_InterBase := 8,
    ODBC_Firebird  := 9,
    XML             := 10,
    OCI_Oracle     := 11,
    NET_MySQL      := 12,
    AzureSQL       := 13,
}
Requirements

Development environment | Target platform | PLC libraries to be linked
---|---|---
TwinCAT v3.1 Build 4020.10 | PC or CX (x86) | Tc3_Database

6.1.2.1.2 E_DBAuthentication

Syntax

Definition:

```plaintext
{attribute 'qualified_only'}
TYPE E_DBAuthentication:
{
    None := 0,
    UserNamePassword := 1,
    x509Cert := 2,
    GSSAPI := 3,
    LDAP := 4
};
END_TYPE
```

Requirements

Development environment | Target platform | PLC libraries to be linked
---|---|---
TwinCAT v3.1 Build 4020.10 | PC or CX (x86) | Tc3_Database

6.1.2.1.3 E_OdbcSubType

Syntax

Definition:

```plaintext
{attribute 'qualified_only'}
TYPE E_OdbcSubType:
{
    Unknown := 0,
    MySQL := 1,
    Oracle := 2,
    Postgre := 3,
    DB2 := 4
    Firebird := 5
};
END_TYPE
```

Requirements

Development environment | Target platform | PLC libraries to be linked
---|---|---
TwinCAT v3.1 Build 4020.10 | PC or CX (x86) | Tc3_Database
6.1.2.1.4  E_TcDBSrvConfigType

Syntax

Definition:

```plaintext
{attribute 'qualified_only'}
TYPE E_TcDBSrvConfigType : 
  (  
    Database := 0,  
    AutoLogGroup := 1,  
    DBSrvSettings := 2,  
    Symbol := 3,  
    ADSDevice := 4,  
    Table := 5,  
    SymbolList := 6  
  );
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.1.5  ST_ConfigAutoLogGrp

This structure is used for the Read method of the function block FB_ConfigTcDBSrvEvt [152]. All configured AutoLog groups are read into the PLC in an array of this structure.

Syntax

Definition:

```plaintext
TYPE ST_ConfigAutoLogGrp : 
  STRUCT  
    sName: T_MaxString;  
    hAutoLogGrpID: UDINT;  
    hDBID: UDINT;  
    sTableName: T_MaxString;  
    stADSDev: ST_ADSDevice;  
    eWriteMode: E_WriteMode;  
    nCycleTime: TIME;  
    nSymbolCount: UDINT;  
  END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sName</td>
<td>T_MaxString</td>
<td>Group name</td>
</tr>
<tr>
<td>hAutoLogGrpID</td>
<td>UDINT</td>
<td>ID of the declared AutoLog group</td>
</tr>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>ID of the assigned database</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Table name</td>
</tr>
<tr>
<td>stADSDev</td>
<td>ST_ADSDevice</td>
<td>ADS device information</td>
</tr>
<tr>
<td>eWriteMode</td>
<td>E_WriteMode</td>
<td>Write mode</td>
</tr>
<tr>
<td>nCycleTime</td>
<td>TIME</td>
<td>Cycle time</td>
</tr>
<tr>
<td>nSymbolCount</td>
<td>UDINT</td>
<td>Number of symbols</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
6.1.2.1.6  ST_ConfigDB

This structure is used for the Read method of the function block `FB_ConfigTcDBsrvEvt` [152]. All configured database connections are read into the PLC in an array of this structure.

**Syntax**

**Definition:**

```plaintext
TYPE ST_ConfigDB :
  STRUCT
    sName: T_MaxString;
    hDBID: DINT;
    eDBType: E_DatabaseType;
    sServer: STRING(80);
    sDatabase: STRING(80);
    bTemp: BOOL;
  END_STRUCT
END_TYPE
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>sName</td>
<td>T_MaxString</td>
<td>Connection name</td>
</tr>
<tr>
<td>hDBID</td>
<td>DINT</td>
<td>ID of the declared database</td>
</tr>
<tr>
<td>eDBType</td>
<td>E_DatabaseType</td>
<td>Database type</td>
</tr>
<tr>
<td>sServer</td>
<td>STRING (80)</td>
<td>Server name</td>
</tr>
<tr>
<td>sDatabase</td>
<td>STRING (80)</td>
<td>Database name</td>
</tr>
<tr>
<td>bTemp</td>
<td>BOOL</td>
<td>TRUE if the connection was only stored temporarily.</td>
</tr>
</tbody>
</table>

**Requirements**

**Development environment**  
TwinCAT v3.1 Build 4020.10

**Target platform**  
PC or CX (x86)

**PLC libraries to be linked**  
Tc3_Database

6.1.2.1.7  ST_ConnStringParameter

**Syntax**

**Definition:**

```plaintext
TYPE ST_ConnStringParameter
  STRUCT
    sName: T_MaxString;
    sValue: T_MaxString;
  END_STRUCT
END_TYPE
```

**Requirements**

**Development environment**  
TwinCAT v3.1 Build 4020.10

**Target platform**  
PC or CX (x86)

**PLC libraries to be linked**  
Tc3_Database

6.1.2.1.8  ConfigType

Different database configuration structures are provided for the supported database types.

**T_DBConfig_ASCII**

Describes the database configuration structure for the ASCII file.
Syntactic Definition:

```plaintext
TYPE T_DBConfig_ASCII
STRUCT
  sServer: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Path to the ASCII file</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_DBConfig_MSAccess**

Describes the database configuration structure for a Microsoft Access database.

Syntactic Definition:

```plaintext
TYPE T_DBConfig_MSAccess
STRUCT
  sServer: T_MaxString;
  sProvider: T_MaxString;
END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Path to the Access databases</td>
</tr>
</tbody>
</table>
Access 2007:  “Microsoft.ACE.OLEDB.12.0” |

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_DBConfig_MsCompactSQL**

Describes the database configuration structure for a Microsoft Compact SQL database.

Syntactic Definition:

```plaintext
TYPE T_DBConfig_MsCompactSQL
STRUCT
  sServer: T_MaxString;
  sPassword: T_MaxString;
  bAuthentification: BOOL;
END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
### PLC API

#### Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Path to the Microsoft Compact SQL file (*.sdf)</td>
</tr>
<tr>
<td>sPassword</td>
<td>T_MaxString</td>
<td>Password for the database</td>
</tr>
<tr>
<td>bAuthentication</td>
<td>BOOL</td>
<td>TRUE if the database is password-protected.</td>
</tr>
</tbody>
</table>

#### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

#### T_DBCConfig_MsExcel

Describes the database configuration structure for a Microsoft Excel file.

**Syntax**

**Definition:**

```
TYPE T_DBCConfig_MsExcel
STRUCT
   sServer: T_MaxString;
   sProvider: T_MaxString;
END_STRUCT
END_TYPE
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Path to the Microsoft Excel file</td>
</tr>
<tr>
<td>sProvider</td>
<td>T_MaxString</td>
<td>&quot;Microsoft.Jet.OLEDB.4.0&quot; or &quot;Microsoft.ACE.OLEDB.12.0&quot;</td>
</tr>
</tbody>
</table>

#### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

#### T_DBCConfig_MsSQL

Describes the database configuration structure for a Microsoft SQL database.

**Syntax**

**Definition:**

```
TYPE T_DBCConfig_MsSQL
STRUCT
   sServer: T_MaxString;
   sProvider: T_MaxString;
   sDatabase: T_MaxString;
   sUserID: T_MaxString;
   sPassword: T_MaxString;
   bAuthentication: BOOL;
END_STRUCT
END_TYPE
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Path to the Microsoft SQL database</td>
</tr>
<tr>
<td>sProvider</td>
<td>T_MaxString</td>
<td>&quot;Microsoft.Jet.OLEDB.4.0&quot; or &quot;Microsoft.ACE.OLEDB.12.0&quot;</td>
</tr>
<tr>
<td>sDatabase</td>
<td>T_MaxString</td>
<td>Database name</td>
</tr>
<tr>
<td>sUserID</td>
<td>T_MaxString</td>
<td>User ID</td>
</tr>
<tr>
<td>sPassword</td>
<td>T_MaxString</td>
<td>Password</td>
</tr>
<tr>
<td>bAuthentication</td>
<td>BOOL</td>
<td>TRUE if the database is password-protected.</td>
</tr>
</tbody>
</table>
Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Enter the name of your SQL server here. Example: &quot;TESTSERVER\SQLEXPRESS&quot;</td>
</tr>
<tr>
<td>sProvider</td>
<td>T_MaxString</td>
<td>“SQLOLEDB” or the provider of the SQL native client, e.g. “SQLNCLI10”</td>
</tr>
<tr>
<td>Database</td>
<td>T_MaxString</td>
<td>Enter the desired database name here.</td>
</tr>
<tr>
<td>sUserID</td>
<td>T_MaxString</td>
<td>Enter the user name here.</td>
</tr>
<tr>
<td>sPassword</td>
<td>T_MaxString</td>
<td>Enter the user password here.</td>
</tr>
<tr>
<td>bAuthentification</td>
<td>BOOL</td>
<td>Set this variable to TRUE to activate authentication based on user ID and password.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_DBConfig_Odbc**

Describes the database configuration structure for a database with ODBC interface.

Syntax

Definition:
```plaintext
TYPE T_DBConfig_NET_MySQL
  STRUCT
    eOdbcSubType: E_OdbcSubType
    nParameterCount: UINT;
    arrParameter: ARRAY [0..MAX_CONFIGPARAMETER] OF ST_ConnStringParameter;
  END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eOdbcSubType</td>
<td>E_OdbcSubType [223]</td>
<td>Describes an ODBC database [223] that is supported with full functionality.</td>
</tr>
<tr>
<td>nParameterCount</td>
<td>UINT</td>
<td>Number of parameters for the connection strings</td>
</tr>
<tr>
<td>arrParameter</td>
<td>ARRAY [0..MAX_CONFIGPARAMETER] OF ST_ConnStringParameter [225];</td>
<td>Array of parameters for the connection string of type ST_ConnStringParameter [225].</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_DBConfig_SQLite**

Describes the database configuration structure for an SQLite database.
Syntax

Definition:

```plaintext
TYPE T_DBConfig_SQLite
  STRUCT
    sServer: T_MaxString;
    sPassword: T_MaxString;
    bAuthentification: BOOL;
  END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Path to the SQLite file</td>
</tr>
<tr>
<td>sPassword</td>
<td>T_MaxString</td>
<td>Password for the database</td>
</tr>
<tr>
<td>bAuthentification</td>
<td>BOOL</td>
<td>TRUE if the database is password-protected.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_DBConfig_XML**

Describes the database configuration structure for an XML file in the customized database format.

Syntax

Definition:

```plaintext
TYPE T_DBConfig_XML
  STRUCT
    sServer: T_MaxString;
    sSchema: T_MaxString;
    sDatabase: T_MaxString;
  END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sServer</td>
<td>T_MaxString</td>
<td>Name and path of the XML file</td>
</tr>
<tr>
<td>sSchema</td>
<td>T_MaxString</td>
<td>Name and path of the XSD file</td>
</tr>
<tr>
<td>sDatabase</td>
<td>T_MaxString</td>
<td>Describes the name of the database. The XML, XSD and XSL files are created automatically for this database type.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
6.1.2.2  NoSQL

6.1.2.2.1  E_NoSQLDatabaseType

Syntax

```plaintext
{attribute 'qualified_only'}
TYPE E_NoSQLDatabaseType :
{
    UnknownType := 0,
    DocumentDB := 1,
    Binary := 2,
    TimeSeriesDB := 3
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.2.2  E_DocumentDBQueryType

Syntax

```plaintext
{attribute 'qualified_only'}
TYPE E_DocumentDbQueryType :
{
    InsertOne := 1,
    InsertMany := 2,
    UpdateOne := 3,
    UpdateMany := 4,
    Find := 5,
    Aggregation := 6,
    Delete := 7,
    DeleteMany := 8,
    CreateCollection := 9,
    DropCollection := 10
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.2.3  T_ObjectId_MongoDB

A data type that maps the database-specific “ObjectId” data type. It consists of 12 bytes with different meanings:

| Table 1: ObjectId at byte level |
|---------------------------------
| 1 2 3 4 5 6 7 8 9 10 11 12 |
| Timestamp MachineId ProcessId Id |

The function block FB_NoSQLObjectId_MongoDB [214] is available for parsing out the individual elements.

Syntax

```plaintext
TYPE T_ObjectId_MongoDB : ARRAY[0..11] OF BYTE;
END_TYPE
```
## Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

## 6.1.2.4 QueryOptions

### DocumentDB

#### T_QueryOptionDocumentDB_Find

**Syntax**

**Definition:**

```c
TYPE T_QueryOptionDocumentDB_Find
STRUCT
  pFilter: POINTER TO BYTE;
  cbFilter: UDINT;
  pSort: POINTER TO BYTE;
  cbSort: UDINT;
  pProjection: POINTER TO BYTE;
  cbProjection: UDINT;
END_STRUCT
END_TYPE
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFilter</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address of the search filter based on which the collection is to be searched.</td>
</tr>
<tr>
<td>cbFilter</td>
<td>UDINT</td>
<td>Length of the search filter.</td>
</tr>
<tr>
<td>pSort</td>
<td>POINTER TO BYTE</td>
<td>Specifies the sort address based on which the collection is to be sorted.</td>
</tr>
<tr>
<td>cbSort</td>
<td>UDINT</td>
<td>Length of sorting</td>
</tr>
<tr>
<td>pProjection</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address of the display and how the collection data is to be displayed.</td>
</tr>
<tr>
<td>cbProjection</td>
<td>UDINT</td>
<td>Length of the display</td>
</tr>
</tbody>
</table>

### QueryOptionDocumentDB_Aggregate

#### Syntax

**Definition:**

```c
TYPE T_QueryOptionDocumentDB_Aggregate
STRUCT
  pPipeStages: POINTER TO BYTE;
  cbPipeStages: UDINT;
END_STRUCT
END_TYPE
```
Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pPipeStages</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address of the stage document. Several stages can be defined in this.</td>
</tr>
<tr>
<td>cbPipeStages</td>
<td>UDINT</td>
<td>Length of the stage document.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_QueryOptionDocumentDB_Insert**

Syntax

Definition:

```c
TYPE T_QueryOptionDocumentDB_Insert
STRUCT
   pDocuments: POINTER TO BYTE;
   cbDocuments: UDINT;
END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDocuments</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address of one or more documents to add to a collection.</td>
</tr>
<tr>
<td>cbDocuments</td>
<td>UDINT</td>
<td>Length of the document.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_QueryOptionDocumentDB_Update**

Syntax

Definition:

```c
TYPE T_QueryOptionDocumentDB_Update
STRUCT
   pFilter: POINTER TO BYTE;
   cbFilter: UDINT;
   pDocuments: POINTER TO BYTE;
   cbDocuments: UDINT;
END_STRUCT
END_TYPE
```
Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFilter</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address of the search filter based on which the collection is to be searched.</td>
</tr>
<tr>
<td>cbFilter</td>
<td>UDINT</td>
<td>Length of the search filter.</td>
</tr>
<tr>
<td>pDocuments</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address of the documents whose values are to be transferred to the collection.</td>
</tr>
<tr>
<td>cbDocuments</td>
<td>UDINT</td>
<td>Length of the documents</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_QueryOptionDocumentDB_Delete**

Syntax

Definition:

```plaintext
TYPE T_QueryOptionDocumentDB_Delete
STRUCT
    pFilter: POINTER TO BYTE;
    cbFilter: UDINT;
END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pFilter</td>
<td>POINTER TO BYTE</td>
<td>Specifies the address of the search filter based on which the collection is to be searched.</td>
</tr>
<tr>
<td>cbFilter</td>
<td>UDINT</td>
<td>Length of the search filter.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**TimeSeriesDB**

**T_QueryOptionTimeSeriesDB_Insert**

This structure is used to write data to a time series database. An array of a structure can be specified as a symbol. Each individual array element is mapped as a row with a unique timestamp in the database. The timestamps of the individual series are generated by specifying a start timestamp and the time cycle between the data sets.

Syntax

Definition:

```plaintext
TYPE T_QueryOptionTimeSeriesDB_Insert
STRUCT
    pSymbol : PVOID;
    cbSymbol : UDINT;
    sDataType : STRING;
    nDataCount : UDINT := 1;
END_STRUCT
```

BECKHOFF
Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSymbol</td>
<td>POINTER TO BYTE</td>
<td>Pointer to the symbol that is to be written to the database</td>
</tr>
<tr>
<td>cbSymbol</td>
<td>UDINT</td>
<td>Length of the specified symbol pointer</td>
</tr>
<tr>
<td>sDataType</td>
<td>STRING</td>
<td>(Simple) type designation of the specified symbol</td>
</tr>
<tr>
<td>nDataCount</td>
<td>UDINT</td>
<td>Number of data sets in the transferred array symbol</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table or table equivalent</td>
</tr>
<tr>
<td>nCycleTime</td>
<td>UDINT</td>
<td>Time interval of the cyclic data sets</td>
</tr>
<tr>
<td>nStartTimestamp</td>
<td>ULINT</td>
<td>Time stamp of the first data set in TwinCAT time: Number of 100 ns intervals since January 1, 1601. Further: Reading the current TwinCAT time</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**T_QueryOptionTimeSeriesDB_Query**

This structure is used to read data from a time series database. The query is passed as a string and can then be passed to the `FB_NoSQLQueryBuilder_TimeSeriesDB`.

Syntax

Definition:

```plaintext
TYPE T_QueryOptionDocumentDB_Find
STRUCT
   pQuery: POINTER TO BYTE;
   cbQuery: UDINT;
END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pQuery</td>
<td>POINTER TO BYTE</td>
<td>Specifies the pointer to the query (string)</td>
</tr>
<tr>
<td>cbQuery</td>
<td>UDINT</td>
<td>Specifies the length of the query pointer</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4022.20</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
6.1.2.5  E_TimeSeriesDbQueryType

Syntax

```plaintext
{attribute 'qualified_only'}
TYPE E_TimeSeriesDbQueryType:
{
  Insert := 1,
  Query := 2
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.3  SQL

6.1.2.3.1  E_ColumnType

Syntax

```plaintext
{attribute 'qualified_only'}
TYPE E_ColumnType:
{
  BigInt := 0,
  Integer := 1,
  SmallInt := 2,
  TinyInt := 3,
  BIT_ := 4,
  Money := 5,
  Float := 6,
  REAL_ := 7,
  DateTime := 8,
  NText := 9,
  NChar := 10,
  Image := 11,
  NVarChar := 12,
  Binary := 13,
  VarBinary := 14
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.3.2  E_SPPParameterType

Syntax

```plaintext
{attribute 'qualified_only'}
TYPE E_SPPParameterType:
{
  Input := 0,
  Output := 1,
  InputOutput := 2,
  ReturnValue := 3,
};
END_TYPE
```
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.3.3 E_VerifyResult

Syntax

Definition:

```
{attribute 'qualified_only'}
TYPE E_VerifyResult:
{
    check_None := 0,
    check_OK := 1,
    check_Error := 2,
    check_TypeWarning := 3,
    check_TypeError := 4,
    check_DataLengthWarning := 5,
    check_DataLengthError := 6
};
END_TYPE
```

Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_None</td>
<td>PLC structure is not verified.</td>
</tr>
<tr>
<td>check_OK</td>
<td>No differences between PLC structure and database table structure</td>
</tr>
<tr>
<td>check_TypeWarning</td>
<td>Different data types with regard to the sign between PLC and database data types, e.g. UINT &lt;&gt; INT</td>
</tr>
<tr>
<td>check_TypeError</td>
<td>Different data types with regard to the sign between PLC structure and database table structure</td>
</tr>
<tr>
<td>check_DataLengthWarning</td>
<td>Different length between PLC structure and database table structure. The record can nevertheless be mapped, although data may be lost.</td>
</tr>
<tr>
<td>check_DataLengthError</td>
<td>Different length between PLC structure and database table structure. The record cannot be mapped.</td>
</tr>
</tbody>
</table>

6.1.2.3.4 ST_SQLSPParameter

This structure is required for the function block FB_SQLStoredProcedureEvt [198] to describe the different parameters of the procedure to be executed.

Syntax

Definition:

```
TYPE ST_SQLSPParameter :
STRUCT
    eParameterType: E_SPParameterType;
    eParameterDataType: E_ColumnType;
    nParameterSize: UDINT;
    sParameterName: STRING(50);
END_STRUCT
END_TYPE
```
### PLC API

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eParameterType</td>
<td>E_SPPParameterType [235]</td>
<td>Parameter type (INPUT, OUTPUT, ..)</td>
</tr>
<tr>
<td>eParameterDataType</td>
<td>E_ColumnType [235]</td>
<td>Parameter type</td>
</tr>
<tr>
<td>nParameterSize</td>
<td>UDINT</td>
<td>Parameter length</td>
</tr>
<tr>
<td>sParameterName</td>
<td>STRING (50)</td>
<td>Parameter name</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

### 6.1.2.4 PLC

#### 6.1.2.4.1 E_ADSRdWrtMode

**Syntax**

**Definition:**

```plaintext
{attribute 'qualified_only'}
TYPE E_ADSRdWrtMode :
{ 
    bySymbolName := 1,
    IGroup_IOffset := 2 
};
END_TYPE
```

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

### 6.1.2.4.2 E_ErrorType

**Syntax**

**Definition:**

```plaintext
{attribute 'qualified_only'}
TYPE E_ErrorType :
{ 
    noError := 0,
    InternalError,
    DataBaseError,
    ADSError 
};
END_TYPE
```

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
6.1.2.4.3 E_ExpParameterType

Syntax

Definition:

```pl
{attribute 'qualified_only'}
TYPE E_ExpParameterType :
{
    NULL := 0,
    Boolean := 1,
    Byte_ := 2,
    Int16 := 3,
    Int32 := 4,
    Int64 := 5,
    UInt16 := 6,
    UInt32 := 7,
    UInt64 := 8,
    DateTime := 9,
    Float32 := 10,
    Double64 := 11,
    STRING_ := 12,
    ByteArray := 13,
    Struct_ := 14,
    XMLTAGName := 15
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.4 E_OrderColumn

Syntax

Definition:

```pl
{attribute 'qualified_only'}
TYPE E_OrderColumn :
{
    ID := 0,
    Timestamp := 1,
    Name := 2,
    Value := 3
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.5 E_OrderType

Syntax

Definition:

```pl
{attribute 'qualified_only'}
TYPE E_OrderType :
{
    ASC := 0,
    DESC := 1
};
END_TYPE
```
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.6 E_PLCDataType

Syntax

Definition:

```plaintext
{attribute 'qualified_only'}
TYPE E_PLCDataType :
{    eType_BOOL  := 0,
    eType_BYTE  := 1,
    eType_SINT  := 2,
    eType_INT   := 3,
    eType_DINT  := 4,
    eType_UINT  := 5,
    eType_UDINT := 6,
    eType_WORD  := 7,
    eType_DWORD := 8,
    eType_LREAL := 9,
    eType_REAL  := 10,
    eType_LINT  := 11,
    eType_ULINT := 12,
    eType_BigType := 13
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.7 E_WriteMode

Syntax

Definition:

```plaintext
{attribute 'qualified_only'}
TYPE E_WriteMode :
{    eADS_TO_DB_Update := 0,
    eADS_TO_DB_Append := 1,
    eADS_TO_DB_RingBuff_Time := 2,
    eADS_TO_DB_RingBuff_Count := 3
};
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.8 ST_ADSDevice

Describes the ADS device, which has to be specified for methods of the function block `FB_PLCDBWriteEvt` [›175].
### Syntax

**Definition:**

```plaintext
TYPE ST_ADSDevice : 
  STRUCT 
    sDevNetID: T_AmsNetId; 
    nDevPort: T_AmsPort; 
    eADSRdWrtMode: E_ADSRdWrtMode; 
    tTimeout: TIME; 
  END_STRUCT 
END_TYPE 
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sDevNetID</td>
<td>T_AmsNetId</td>
<td>NetID of the ADS device</td>
</tr>
<tr>
<td>nDevPort</td>
<td>T_AmsPort</td>
<td>AMS Port</td>
</tr>
<tr>
<td>eADSRdWrtMode</td>
<td>E_ADSRdWrtMode</td>
<td>Connection mode IGroup_IOffset / bySymbol</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>ADS connection timeout</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

### 6.1.2.4.9 ST_AutoLogGrpStatus

Provides information about the respective AutoLog group.

**Syntax**

**Definition:**

```plaintext
TYPE ST_AutoLogGrpStatus : 
  STRUCT 
    hAutoLogGrpID: UDINT; 
    nCycleCount: UDINT; 
    hrErrorCode: HRESULT; 
    eErrorType: E_ErrorType; 
    bError: BOOL; 
  END_STRUCT 
END_TYPE 
```

**Parameter**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAutoLogGrpID</td>
<td>UDINT</td>
<td>ID of the declared AutoLog group</td>
</tr>
<tr>
<td>nCycleCount</td>
<td>UDINT</td>
<td>Number of executed cycles</td>
</tr>
<tr>
<td>hrErrorCode</td>
<td>HRESULT</td>
<td>HRESULT error code</td>
</tr>
<tr>
<td>eErrorType</td>
<td>E_ErrorType</td>
<td>Error type</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE if an error has occurred.</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
6.1.2.4.10  ST_ColmnInfo

Syntax

Definition:

```plaintext
TYPE ST_ColmnInfo : 
  STRUCT
    sName: STRING(50);
    sProperty: STRING;
    nLength: UDINT;
    eType: EColumnType;
  END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sName</td>
<td>STRING (50)</td>
<td>Name of the column</td>
</tr>
<tr>
<td>sProperty</td>
<td>STRING</td>
<td>String for additional column properties</td>
</tr>
<tr>
<td>nLength</td>
<td>UDINT</td>
<td>Maximum length (for strings and byte streams)</td>
</tr>
<tr>
<td>eType</td>
<td>EColumnType</td>
<td>Column type</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.11  ST_ExpParameter

This structure is required for the function block FB PLCCmd [180], for making the description of the different parameters (placeholders) available in the SQL command.

Syntax

Definition:

```plaintext
TYPE ST_ExpParameter : 
  STRUCT
    sParaName : T_MaxString;
    nParaSize : UDINT;
    eParaType : E_ExpParameterType;
  END_STRUCT
END_TYPE
```

Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sParaName</td>
<td>T_MaxString</td>
<td>Name of the parameter (placeholder)</td>
</tr>
<tr>
<td>nParaSize</td>
<td>UDINT</td>
<td>Length of the parameter value</td>
</tr>
<tr>
<td>eParaType</td>
<td>E_ExpParameterType [238]</td>
<td>Data type of the parameter</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
6.1.2.4.12 ST_StandardRecord

This structure can be used in the PLC if you want to work with the standard table structure of the TwinCAT Database Server.

This structure cannot be used with Microsoft Access databases, since this database type does not support the 64-bit integer data type. In this case the structure `ST_StandardRecord_MSAccess` should be used.

Syntax

Definition:

```plaintext
TYPE ST_StandardRecord :
  STRUCT
    nID: LINT;
    dtTimestamp: DT;
    sName: STRING(80);
    rValue: LREAL;
  END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.13 ST_StandardRecord_MSAccess

This structure can be used in the PLC if you want to work with the standard table structure of the TwinCAT Database Server. This structure is specifically intended for Microsoft Access databases, since this database type does not support the 64-bit integer data type.

Syntax

Definition:

```plaintext
TYPE ST_StandardRecord_MSAccess :
  STRUCT
    nID: DINT;
    dtTimestamp: DT;
    sName: STRING(80);
    rValue: LREAL;
  END_STRUCT
END_TYPE
```

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.2.4.14 ST_Symbol

Describes the ADS symbol, which has to be specified for methods of the function block `FB_PLCDBWriteEvt`.

Syntax

Definition:

```plaintext
TYPE ST_Symbol :
  STRUCT
    sSymbolName: T_MaxString;
    sDBSymbolName: T_MaxString;
    nIGroup: UDINT;
    nIOffset: UDINT;
  END_STRUCT
END_TYPE
```

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>
Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSymbolName</td>
<td>T_MaxString</td>
<td>Symbol name</td>
</tr>
<tr>
<td>sDBSymbolName</td>
<td>T_MaxString</td>
<td>Name to be written to the database.</td>
</tr>
<tr>
<td>nIGroup</td>
<td>UDINT</td>
<td>Index Group (only for ADSRdWrtMode &quot;eADSMode_IGroup_IOffset&quot;)</td>
</tr>
<tr>
<td>nIOffset</td>
<td>UDINT</td>
<td>Index Offset (only for ADSRdWrtMode &quot;eADSMode_IGroup_IOffset&quot;)</td>
</tr>
<tr>
<td>nBitSize</td>
<td>UDINT</td>
<td>Length in bits (only for ADSRdWrtMode &quot;eADSMode_IGroup_IOffset&quot;)</td>
</tr>
<tr>
<td>eDataType</td>
<td>E_PLCDataType</td>
<td>Data type (only for ADSRdWrtMode &quot;eADSMode_IGroup_IOffset&quot;)</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.3 Global constants

6.1.3.1 Constants

VAR_GLOBAL CONSTANT GVL

AMS.PORT_DBSRV : UINT := 21372;
MAX_DBCONNECTIONS : UDINT := 255;
MAX_DBCOLUMNS : UDINT := 255;
MAX_SPPARAMETER : UDINT := 255;
MAX_CONFIGURATIONS : UDINT := 255;
MAX_RECORDS : UDINT := 255;

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

6.1.4 Obsolete

6.1.4.1 Configure mode

6.1.4.1.1 FB_ConfigTcDBSrv

Function block for creating, reading and deleting configuration entries for the TwinCAT Database Server.
**Syntax**

**Definition:**

```
FUNCTION_BLOCK FB_ConfigTcDBSrv
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Local</td>
<td>Creates new entries in the XML configuration file for the TwinCAT Database Server</td>
</tr>
<tr>
<td>Read</td>
<td>Local</td>
<td>Reads the current configuration of the TwinCAT Database Server</td>
</tr>
<tr>
<td>Delete</td>
<td>Local</td>
<td>Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Create**

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

**Syntax**

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

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

Creating AutoLog groups is currently not supported.

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

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

Read

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

Syntax

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDBConfig</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB</td>
<td>Pointer address of the array into which the database configurations are to be written.</td>
</tr>
<tr>
<td>cbDBConfig</td>
<td>UDINT</td>
<td>Length of the database configuration array</td>
</tr>
<tr>
<td>pAutoLogGrpConfig</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp</td>
<td>Pointer address of the array into which the AutoLogGrp configurations are to be written.</td>
</tr>
<tr>
<td>cbAutoLogGrpConfig</td>
<td>UDINT</td>
<td>Length of the AutoLogGrp configuration array</td>
</tr>
<tr>
<td>pDBCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of database configurations.</td>
</tr>
<tr>
<td>pAutoLogGrpCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of AutoLogGrp configurations.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

**Delete**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTcDBSrvConfigType</td>
<td>E_TcDBSrvConfigType</td>
<td>Type of the configuration to be deleted (database / AutoLog group)</td>
</tr>
<tr>
<td>hConfigID</td>
<td>UDINT</td>
<td>ID of the configuration to be deleted (hDBID or hAutoLogGrpID)</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

## Sample

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

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

## 6.1.4.1.2 FB_PLCDBAutoLog

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

### Syntax

#### Definition:

```
FUNCTION_BLOCK FB_PLCDBAutoLog
VAR_INPUT
  sNetID: T_AmsNetID := '':
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
  bBusy_Status: BOOL;
END_VAR
```

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>
## Outputs

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

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>Local</td>
<td>Executes the AutoLog group once</td>
</tr>
<tr>
<td>Start</td>
<td>Local</td>
<td>Starts AutoLog mode with the corresponding configured AutoLog groups</td>
</tr>
<tr>
<td>Status</td>
<td>Local</td>
<td>Queries the status of the AutoLog groups.</td>
</tr>
<tr>
<td>Stop</td>
<td>Local</td>
<td>Stops AutoLog mode</td>
</tr>
</tbody>
</table>

## Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

## RunOnce

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

### Syntax

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

## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAutoLogGrpID</td>
<td>UDINT</td>
<td>ID of the AutoLog group to be executed once.</td>
</tr>
<tr>
<td>bAll</td>
<td>BOOL</td>
<td>If TRUE, all AutoLog groups are executed once.</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>
Sample

```plaintext
VAR
  fbPLCDBAutoLog : FB_PLCDBAutoLog (sNetID='', tTimeout := T#5S);
END_VAR
IF fbPLCDBAutoLog.RunOnce(hAutologGrpID := 1, bAll := FALSE) THEN
  // ...
END_IF
```

### Start

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

### Syntax

```plaintext
METHOD Start : BOOL
```

#### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

```plaintext
VAR
  fbPLCDBAutoLog : FB_PLCDBAutoLog (sNetID='', tTimeout := T#5S);
END_VAR
IF fbPLCDBAutoLog.Start() THEN
  // ...
END_IF
```

### Status

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

### Syntax

```plaintext
METHOD Status : BOOL
```

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCheckCycle</td>
<td>TIME</td>
<td>Interval time at which the status array is updated.</td>
</tr>
<tr>
<td>pError</td>
<td>POINTER TO BOOL</td>
<td>TRUE, if an error has occurred in AutoLog mode.</td>
</tr>
<tr>
<td>pAutoLogStatus</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus</td>
<td>Address of the status array that contains all groups.</td>
</tr>
<tr>
<td>cbAutoLogStatus</td>
<td>UDINT</td>
<td>Length of the status array.</td>
</tr>
</tbody>
</table>

```
Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

Stop

This method stops the AutoLog mode.

Syntax

METHOD Stop : BOOL

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

6.1.4.2 PLC Expert mode

6.1.4.2.1 FB_ConfigTcDBSrv

Function block for creating, reading and deleting configuration entries for the TwinCAT Database Server.

Syntax

Definition:

```java
FUNCTION_BLOCK FB_ConfigTcDBSrv
VAR_INPUT
    sNetID : T_AmsNetID := ';
    tTimeout : TIME := T#5S;
END_VAR
VAR_OUTPUT
```

```java
```

Stop

This method stops the AutoLog mode.

Syntax

METHOD Stop : BOOL

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Local</td>
<td>Creates new entries in the XML configuration file for the TwinCAT Database Server.</td>
</tr>
<tr>
<td>Read</td>
<td>Local</td>
<td>Reads the current configuration of the TwinCAT Database Server.</td>
</tr>
<tr>
<td>Delete</td>
<td>Local</td>
<td>Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Create**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pTcDBSrvConfig</td>
<td>POINTER TO BYTE</td>
<td>Pointer of the configuration structure to be created.</td>
</tr>
<tr>
<td>cbTcDBSrvConfig</td>
<td>UDINT</td>
<td>Length of the configuration structure</td>
</tr>
<tr>
<td>bTemporary</td>
<td>BOOL</td>
<td>Indicates whether the configuration is to be stored in the XML file.</td>
</tr>
<tr>
<td>pConfigID</td>
<td>POINTER TO UDINT</td>
<td>Return pointer of the configuration ID (hDBID or hAutoLogGrpId)</td>
</tr>
</tbody>
</table>

Creating AutoLog groups is currently not supported.

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

### Sample

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

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

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

## Read

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

### Syntax

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDBConfig</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB</td>
<td>Pointer address of the array into which the database configurations are to be written.</td>
</tr>
<tr>
<td>cbDBConfig</td>
<td>UDINT</td>
<td>Length of the database configuration array</td>
</tr>
<tr>
<td>pAutoLogGrpConfig</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp</td>
<td>Pointer address of the array into which the AutoLogGrp configurations are to be written.</td>
</tr>
<tr>
<td>cbAutoLogGrpConfig</td>
<td>UDINT</td>
<td>Length of the AutoLogGrp configuration array</td>
</tr>
<tr>
<td>pDBCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of database configurations.</td>
</tr>
<tr>
<td>pAutoLogGrpCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of AutoLogGrp configurations.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

**Delete**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTcDBSrvConfigType</td>
<td>E_TcDBSrvConfigType</td>
<td>Type of the configuration to be deleted (database / AutoLog group)</td>
</tr>
<tr>
<td>hConfigID</td>
<td>UDINT</td>
<td>ID of the configuration to be deleted (hDBID or hAutoLogGrpID)</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

### Sample

```plaintext
VAR
  fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
  myConfigHandle : INT;
END_VAR

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

### 6.1.4.2.2 FB_PLCDBAutoLog

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

#### Syntax

**Definition:**

```plaintext
FUNCTION_BLOCK FB_PLCDBAutoLog
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
  bBusy_Status: BOOL;
END_VAR
```

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>
### Outputs

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

### Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>Local</td>
<td>Executes the AutoLog group once</td>
</tr>
<tr>
<td>Start[158]</td>
<td>Local</td>
<td>Starts AutoLog mode with the corresponding configured AutoLog groups</td>
</tr>
<tr>
<td>Status[158]</td>
<td>Local</td>
<td>Queries the status of the AutoLog groups.</td>
</tr>
<tr>
<td>Stop[159]</td>
<td>Local</td>
<td>Stops AutoLog mode</td>
</tr>
</tbody>
</table>

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

### RunOnce

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

**Syntax**

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

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAutoLogGrpID</td>
<td>UDINT</td>
<td>ID of the AutoLog group to be executed once.</td>
</tr>
<tr>
<td>bAll</td>
<td>BOOL</td>
<td>If TRUE, all AutoLog groups are executed once.</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RunOnce</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>
**Sample**

```plaintext
VAR
  fbPLCDBAutoLog : FB_PLCDBAutoLog (sNetID=' ', tTimeout := T#5S);
END_VAR
IF fbPLCDBAutoLog.RunOnce(hAutologGrpID := 1, bAll := FALSE) THEN
  // ... END_IF
```

**Start**

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

**Syntax**

```plaintext
METHOD Start : BOOL
```

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

```plaintext
VAR
  fbPLCDBAutoLog : FB_PLCDBAutoLog (sNetID=' ', tTimeout := T#5S);
END_VAR
IF fbPLCDBAutoLog.Start() THEN
  // ... END_IF
```

**Status**

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

**Syntax**

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

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tCheckCycle</td>
<td>TIME</td>
<td>Interval time at which the status array is updated.</td>
</tr>
<tr>
<td>pError</td>
<td>POINTER TO BOOL</td>
<td>TRUE, if an error has occurred in AutoLog mode.</td>
</tr>
<tr>
<td>pAutoLogGrpStatus</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_AutoLogGrpStatus</td>
<td>Address of the status array that contains all groups.</td>
</tr>
<tr>
<td>cbAutoLogGrpStatus</td>
<td>UDINT</td>
<td>Length of the status array</td>
</tr>
</tbody>
</table>
### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

#### Sample

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

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

### Stop

This method stops the AutoLog mode.

#### Syntax

```plaintext
METHOD Stop : BOOL
```

#### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

#### Sample

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

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

### 6.1.4.2.3 FB_PLCDBCreate

Function block with two methods. One method can be used to create databases from the PLC on a database server specified in the PLC. The other method can be used to generate a new table in a specified database.

#### Syntax

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

---

TF6420 Version: 1.11
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Local</td>
<td>Creates a new database</td>
</tr>
<tr>
<td>Table</td>
<td>Local</td>
<td>Creates a new table with a structure that is defined via an array with x elements or x columns in the PLC.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Database**

This method creates a new database. Optionally you can specify whether the created database should also be used for the configuration of the TwinCAT Database Server.

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDatabaseConfig</td>
<td>POINTER TO BYTE</td>
<td>Address of the database configuration structure [225]</td>
</tr>
<tr>
<td>cbDatabaseConfig</td>
<td>UDINT</td>
<td>Length of the database configuration structure</td>
</tr>
<tr>
<td>bCreateXMLConfig</td>
<td>BOOL</td>
<td>Indicates whether the newly created database should be entered as new configuration entry in the XML file.</td>
</tr>
<tr>
<td>pDBID</td>
<td>UDINT</td>
<td>Returns the hDBID if/when a new configuration entry was created.</td>
</tr>
</tbody>
</table>

### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

### Sample

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

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

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

### Table

This method creates a new table with a structure that is defined through an array with x elements or x columns in the PLC.

### Syntax

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>MaxString</td>
<td>Name of the table to be created.</td>
</tr>
<tr>
<td>pTableCfg</td>
<td>POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ColumnInfo</td>
<td>Indicates the pointer address of the table structure array. The individual columns are written in this array.</td>
</tr>
<tr>
<td>cbTableCfg</td>
<td>UDINT</td>
<td>Indicates the length of the array in which the columns are configured.</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

## Sample

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

ColumnInfo[0].sName := 'colBigInt'; ColumnInfo[0].eType := E_ColumnType.BigInt; ColumnInfo[0].nLength := 8; ColumnInfo[0].sProperty := 'IDENTITY(1,1)';
ColumnInfo[1].sName := 'colInteger'; ColumnInfo[1].eType := E_ColumnType.Integer; ColumnInfo[1].nLength := 4;
ColumnInfo[2].sName := 'colSmallInt'; ColumnInfo[2].eType := E_ColumnType.SmallInt; ColumnInfo[2].nLength := 2;
ColumnInfo[3].sName := 'colTinyInt'; ColumnInfo[3].eType := E_ColumnType.TinyInt; ColumnInfo[3].nLength := 1;
ColumnInfo[4].sName := 'colBit'; ColumnInfo[4].eType := E_ColumnType.BIT_; ColumnInfo[4].nLength := 1;
ColumnInfo[5].sName := 'colMoney'; ColumnInfo[5].eType := E_ColumnType.Money; ColumnInfo[5].nLength := 8;
ColumnInfo[6].sName := 'colFloat'; ColumnInfo[6].eType := E_ColumnType.Float; ColumnInfo[6].nLength := 8;
ColumnInfo[7].sName := 'colReal'; ColumnInfo[7].eType := E_ColumnType.REAL_; ColumnInfo[7].nLength := 4;
ColumnInfo[8].sName := 'colDate'; ColumnInfo[8].eType := E_ColumnType.DateTime; ColumnInfo[8].nLength := 4;
ColumnInfo[9].sName := 'colTime'; ColumnInfo[9].eType := E_ColumnType.DateTime; ColumnInfo[9].nLength := 8;
ColumnInfo[10].sName := 'colChar'; ColumnInfo[10].eType := E_ColumnType.Character; ColumnInfo[10].nLength := 10;
ColumnInfo[12].sName := 'colVarChar'; ColumnInfo[12].eType := E_ColumnType.VarChar; ColumnInfo[12].nLength := 50;
ColumnInfo[13].sName := 'colBinary'; ColumnInfo[13].eType := E_ColumnType.Binary; ColumnInfo[13].nLength := 30;
ColumnInfo[14].sName := 'colVarBinary'; ColumnInfo[14].eType := E_ColumnType.VarBinary; ColumnInfo[14].nLength := 20;

IF fbPLCDBCreate.Table(hDBID:= 1, sTableName := 'myNewTable', pTableCfg := ADR(ColumnInfo), cbTableCfg := SIZEOF(ColumnInfo)) THEN
  IF fbPLCDBCreate.bError THEN
    TcMessage := fbPLCDBCreate.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF
```
6.1.4.2.4 FB_PLCDBRead

Function block for reading records from a database.

Syntax

FUNCTION_BLOCK FB_PLCDBRead
  VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
  END_VAR
  VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
  END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read [261]</td>
<td>Local</td>
<td>Reads a specified number of records from a database table with the standard table structure specified by Beckhoff.</td>
</tr>
<tr>
<td>ReadStruct [263]</td>
<td>Local</td>
<td>Reads a specified number of records from a database table with any table structure.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Read

This method reads a specified number of records from a database table with the standard table structure specified by Beckhoff. (The standard table structure is used in AutoLog mode and in the FB_DBWrite function block, for example).
Syntax

METHOD Read : BOOL
VAR_INPUT
  hDBID: UDINT;
  sTableName: T_MaxString;
  sDBSymbolName: T_MaxString;
  eOrderBy: E_OrderColumn := E_OrderColumn.eColumnID;
  eOrderType: E_OrderType := E_OrderType.eOrder_ASC;
  nStartIndex: UDINT;
  nRecordCount: UDINT;
  pData: POINTER TO ST_StandardRecord;
  cbData: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>sDBSymbolName</td>
<td>T_MaxString</td>
<td>Symbol name to be read from the standard table  structure.</td>
</tr>
<tr>
<td>eOrderBy</td>
<td>E_OrderColumn.eColumnID</td>
<td>Sorting column (ID, timestamp, name or value)</td>
</tr>
<tr>
<td>eOrderType</td>
<td>E_OrderType.eOrder_ASC</td>
<td>Sorting direction (ASC or DESC)</td>
</tr>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO ST_StandardRecord</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

VAR
  fbPLCDBRead : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
  ReadStruct : ST_StandardRecord;
  tcMessage : I_TcMessage;
END_VAR

IF fbPLCDBRead.Read(
  hDBID:= 1,
  sTableName:= 'MyTable_WithLReal',
  sDBSymbolName:= 'MyValue',
  eOrderBy:= E_OrderColumn.ID,
  eOrderType:= E_OrderType.DESC,
  nStartIndex:= 0,
  nRecordCount:= 1,
  pData:= ADR(ReadStruct),
  cbData:= SIZEOF(ReadStruct))
THEN
  IF fbPLCDBRead.bError THEN
    tcMessage := fbPLCDBRead.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF

Result in the PLC:
ReadStruct

This method reads a specified number of records from a database table with any table structure.

Syntax

METHOD ReadStruct : BOOL
VAR_INPUT
  hDBID: UDINT;
  sTableName: T_MaxString;
  pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
  cbColumnNames: UDINT;
  sOrderByColumn: STRING(50);
  eOrderType: E_OrderType := E_OrderType.eOrder_ASC
  nStartIndex: UDINT;
  nRecordCount: UDINT;
  pData: POINTER TO BYTE;
  cbData: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>pColumnNames</td>
<td>POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)</td>
<td>Address of the array containing the column name to be read.</td>
</tr>
<tr>
<td>cbColumnNames</td>
<td>UDINT</td>
<td>Length of the column name array</td>
</tr>
<tr>
<td>sOrderByColumn</td>
<td>STRING(50)</td>
<td>Name the sorting column</td>
</tr>
<tr>
<td>eOrderType</td>
<td>E_OrderType := E_OrderType.eOrder_ASC</td>
<td>Sorting direction (ASC or DESC)</td>
</tr>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadStruct</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

VAR
  fbPLCDBRead : FB_PLCDBReadEvt (sNetID := '', tTimeout := T#5S);
  myCustomStruct : ST_Record;
  tcMessage : 1_TcMessage;
END_VAR
TYPE ST_Record :
  STRUCT
    nID        : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName      : STRING;
    nSensor1   : LREAL;
    nSensor2   : LREAL;
  END_STRUCT
END_TYPE

// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';

IF fbPLCDBRead.ReadStruct(
    hDBID:= 1,
    sTableName:= 'MyTable_Struct',
    pColumnNames:= ADR(ColumnNames),
    cbColumnNames:= SIZEOF(ColumnNames),
    sOrderByColumn:= ColumnNames[0],
    eOrderType:= E_OrderType.DESC,
    nStartIndex:= 0,
    nRecordCount:= 1,
    pData:= ADR(myCustomStruct),
    cbData:= SIZEOF(myCustomStruct))
THEN
  IF fbPLCDBRead.bError THEN
    tcMessage:= fbPLCDBRead.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF

Result in the PLC:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>myCustomStruct</td>
<td>ST_Record</td>
<td></td>
</tr>
<tr>
<td>nID</td>
<td>LINT</td>
<td>1</td>
</tr>
<tr>
<td>dtTimestamp</td>
<td>DATE_AND_TIME</td>
<td>DT#2018-2-1-15:17:54</td>
</tr>
<tr>
<td>sName</td>
<td>STRING</td>
<td>'MyStructVal'</td>
</tr>
<tr>
<td>nSensor1</td>
<td>LREAL</td>
<td>12.34</td>
</tr>
<tr>
<td>nSensor2</td>
<td>LREAL</td>
<td>102.5</td>
</tr>
</tbody>
</table>

### 6.1.4.2.5 FB_PLCDBWrite

Function block for writing of records into a database.

**Syntax**

**Definition:**

FUNCTION_BLOCK FB_PLCDBWrite
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>iPcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

## Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write [265]</td>
<td>Local</td>
<td>Creates a record in the standard table structure specified by Beckhoff.</td>
</tr>
<tr>
<td>WriteBySymbol</td>
<td>Local</td>
<td>Reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff.</td>
</tr>
<tr>
<td>WriteStruct [268]</td>
<td>Local</td>
<td>Creates a record with any table structure</td>
</tr>
</tbody>
</table>

## Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

## Write

This method creates a record in the standard table structure specified by Beckhoff.

### Syntax

```
METHOD Write : BOOL
VAR_INPUT
    hDBID: UDINT;
    sTableName: T_MaxString;
    pValue: POINTER TO BYTE;
    cbValue: UDINT;
    sDBSymbolName: T_MaxString;
    eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
    nRingBuffParameter: UDINT;
END_VAR
```
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>pValue</td>
<td>POINTER TO BYTE</td>
<td>Address of the variable to be logged in the standard table structure.</td>
</tr>
<tr>
<td>cbValue</td>
<td>UDINT</td>
<td>Length of the variable to be logged.</td>
</tr>
<tr>
<td>sDBSymbolName</td>
<td>T_MaxString</td>
<td>Name that is logged in the table.</td>
</tr>
<tr>
<td>eDBWriteMode</td>
<td>E_WriteMode</td>
<td>Indicates the write mode. (append, update, ring buffer)</td>
</tr>
<tr>
<td>nRingBuffParameter</td>
<td>UDINT</td>
<td>Additional parameter(s) for the &quot;ring buffer&quot; write mode.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

This sample shows how to use the FB_PLCDBWriteEvt.Write method:

```plaintext
VAR
    fbPLCDBWrite : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
    myValue : LREAL := 43.23;
    tcMessage : I_TcMessage;
END_VAR

IF fbPLCDBWrite.Write(
    hDBID:= 1,
    sTableName:= 'myTable_WithLReal',
    pValue:= ADR(myValue),
    cbValue:= SIZEOF(myValue),
    sDBSymbolName:= 'MyValue',
    eDBWriteMode:= E_WriteMode.eADS_TO_DB_RingBuff_Count,
    nRingBuffParameter:= 3)
THEN
    IF fbPLCDBWrite.bError THEN
        tcMessage := fbPLCDBWrite.ipTcResult;
        nState := 255;
    ELSE
        nState := 0;
    END_IF
END_IF
```

**Result in the database:**

<table>
<thead>
<tr>
<th>ID</th>
<th>Timestamp</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>Has been dropped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>'2018-01-30 14:04:19'</td>
<td>'MyValue'</td>
<td>41.23</td>
</tr>
<tr>
<td>29</td>
<td>'2018-01-30 14:04:29'</td>
<td>'MyValue'</td>
<td>42.23</td>
</tr>
<tr>
<td>30</td>
<td>'2018-01-30 14:04:39'</td>
<td>'MyValue'</td>
<td>43.23</td>
</tr>
</tbody>
</table>

With the ring buffer option, only three entries of this name are in the database at any one time. Older entries are deleted.

**WriteBySymbol**

This method reads the value of a specified ADS symbol and saves it in the standard table structure specified by Beckhoff. ADS symbols from other ADS devices can also be read.
Syntax

METHOD WriteBySymbol : BOOL
VAR_INPUT
  hDBID: UDINT;
  sTableName: T_MaxString;
  stADSDevice : ST_ADSDevice;
  stSymbol : ST_Symbol;
  eDBWriteMode: E_WriteMode := E_WriteMode.eADS_TO_DB_Append;
  nRingBuffParameter: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>stADSDevice</td>
<td>ST_ADSDevice</td>
<td>ADS device from which a symbol is to be logged in the standard table structure.</td>
</tr>
<tr>
<td>stSymbol</td>
<td>ST_Symbol</td>
<td>Symbol name of the variable to be written</td>
</tr>
<tr>
<td>eDBWriteMode</td>
<td>E_WriteMode</td>
<td>Indicates the write mode. (append, update, ring buffer)</td>
</tr>
<tr>
<td>nRingBuffParameter</td>
<td>UDINT</td>
<td>Additional parameter(s) for the &quot;ring buffer&quot; write mode</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteBySymbol</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

This sample shows how to use the FB_PLCDBWriteEvt.WriteBySymbol method:

VAR
  fbPLCDBWrite : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
  myValue     := 43.23;
  myAdsDevice := ST_ADSDevice;
  mySymbol    := ST_Symbol;
  tcMessage   := I_TcMessage;
END_VAR

// Set ADSDevice Information
myAdsDevice.sDevNetID := '127.0.0.1.1.1';
myAdsDevice.nDevPort := 851;
myAdsDevice.eADSRdWrtMode := E_ADSRdWrtMode.bySymbolName;
myAdsDevice.tTimeout := T#5S;

// Set Symbol Information
mySymbol.eDataType := E_PLCDataType.eType_LREAL;
mySymbol.sDBSymbolName := 'MySymbol';
mySymbol.sSymbolName := 'MAIN.myValue';
mySymbol.nBitSize := 8;

// Call Functionblock
IF fbPLCDBWrite.WriteBySymbol(
  hDBID:= 1,
  sTableName:= 'myTable_WithLReal',
  stADSDevice:= myAdsDevice,
  stSymbol:= mySymbol,
  eDBWriteMode:= E_WriteMode.eADS_TO_DB_Append,
  nRingBuffParameter:= 1)
THEN
  IF fbPLCDBWrite.bError THEN
    tcMessage := fbPLCDBWrite.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF
Result in the database:

<table>
<thead>
<tr>
<th>ID</th>
<th>Timestamp</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>'2018-01-30 14:04:19'</td>
<td>'MyValue'</td>
<td>41.23</td>
</tr>
<tr>
<td>29</td>
<td>'2018-01-30 14:04:29'</td>
<td>'MyValue'</td>
<td>42.23</td>
</tr>
<tr>
<td>30</td>
<td>'2018-01-30 14:04:39'</td>
<td>'MyValue'</td>
<td>43.23</td>
</tr>
<tr>
<td>31</td>
<td>'2018-01-30 14:06:12'</td>
<td>'MySymbol'</td>
<td>86.2</td>
</tr>
</tbody>
</table>

**WriteStruct**

This method creates a record with a freely selectable table structure.

**Syntax**

```plaintext
METHOD WriteStruct : BOOL
VAR_INPUT
  hDBID: UDINT;
  sTableName: T_MaxString;
  pRecord: POINTER TO BYTE;
  cbRecord: UDINT;
  pColumnNames: POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50);
  cbColumnNames: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
<td>Name of the table that is to be read.</td>
</tr>
<tr>
<td>pRecord</td>
<td>POINTER TO BYTE</td>
<td>Address of a structure that is to be logged in a freely selectable table structure.</td>
</tr>
<tr>
<td>cbRecord</td>
<td>UDINT</td>
<td>Length of the structure to be written</td>
</tr>
<tr>
<td>pColumnNames</td>
<td>POINTER TO ARRAY [0..MAX_DBCOLUMNS] OF STRING(50)</td>
<td>Address of the array containing the column name to be filled.</td>
</tr>
<tr>
<td>cbColumnNames</td>
<td>UDINT</td>
<td>Length of the column name array</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteStruct</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

This sample shows how to use the method FB_PLCDBWriteEvt.WriteStruct:

```plaintext
VAR
  fbPLCDBWrite : FB_PLCDBWriteEvt(sNetID := '', tTimeout := T#5S);
  myRecord     : ST_Record;
  ColumnNames  : ARRAY[0..4] OF STRING(50);
  systime      : GETSYSTEMTIME;
  currentTime  : T_FILETIME;
  tcMessage    : I_TcMessage;
END_VAR

TYPE ST_Record :
  STRUCT
    nID    : LINT;
    dtTimestamp: DATE_AND_TIME;
    sName  : STRING;
    nSensor1 : LREAL;
```
```plaintext
nSensor2 : LREAL;
END_STRUCT
END_TYPE

// set Values
systime(timeLoDw => currentTime.dwLowDateTime, timeHiDW => currentTime.dwHighDateTime);
myRecord.dtTimestamp := FILETIME_TO_DT(currentTime);
myRecord.sName := 'MyStructVal';
myRecord.nSensor1 := 12.34;
myRecord.nSensor2 := 102.5;

// set columnnames
ColumnNames[0] := 'ID';
ColumnNames[1] := 'Timestamp';
ColumnNames[2] := 'Name';

// Call Functionblock
IF fbPLCDBWrite.WriteStruct(
hDBID:= 1,
sTableName:= 'myTable_Struct',
pRecord:= ADR(myRecord),
cbRecord:= SIZEOF(myRecord),
pColumnNames:= ADR(ColumnNames),
cbColumnNames:= SIZEOF(ColumnNames))
THEN
  IF fbPLCDBWrite.bError THEN
    tcMessage := fbPLCDBWrite.ipTcResult;
nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF

Result in the database:

<table>
<thead>
<tr>
<th>ID</th>
<th>Timestamp</th>
<th>Name</th>
<th>Sensor1</th>
<th>Sensor2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>'2018-01-30 15:23:26'</td>
<td>'MyStructVal'</td>
<td>12.34</td>
<td>102.5</td>
</tr>
</tbody>
</table>

6.1.4.2.6 FB_PLCDBCmd

Function block with two methods. Users can define and transfer their own SQL commands. Placeholders in the SQL command can correlate with structures in the PLC, which reflect the table structure. The Database Server ultimately enters the current data of the structure into the SQL command.

Syntax

Definition:

FUNCTION_BLOCK FB_PLCDBCmd
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#$5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Local</td>
<td>Sends any SQL commands to the database. Returned records cannot be read.</td>
</tr>
<tr>
<td>ExecuteDataReturn</td>
<td>Local</td>
<td>Sends any SQL commands to the database. A specified number of records can be read.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Execute

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are then replaced by the TwinCAT Database Server with the corresponding values before the execution. Returned records cannot be read.

Syntax

METHOD Execute : BOOL
VAR_INPUT
    hDBID: UDINT;
    pExpression: POINTER TO BYTE;
    cbExpression: UDINT;
    pData: POINTER TO BYTE;
    cbData: UDINT;
    pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
    cbParameter: UDINT;
END_VAR

Sample

VAR
    fbPLCDBCmd : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
    sCmd : STRING (1000);
    myStruct : ST_DataAll;
    aPara : ARRAY[0..14] OF ST_ExpParameter;
    tcMessage : I_TcMessage;
END_VAR
TYPE ST_DataAll:
  STRUCT
    colBigInt: LINT;
    colInteger: DINT;
    colSmallInt: INT;
    colTinyInt: BYTE;
    colBit: BOOL;
    colMoney: LREAL;
    colFloat: LREAL;
    colReal: REAL;
    colDateTime: DT;
    colNText: STRING(255);
    colNChar: STRING(10);
    colImage: ARRAY[0..255] OF BYTE;
    colNVarChar: STRING(50);
    colBinary: ARRAY[0..29] OF BYTE;
  END_STRUCT
END_TYPE

// set Parameter configuration
aPara[0].sParaName := 'colBigInt';  aPara[0].eParaType := E_ExpParameterType.Int64;  aPara[0].nParaSize := 8;
aPara[1].sParaName := 'colInteger';  aPara[1].eParaType := E_ExpParameterType.Int32;  aPara[1].nParaSize := 4;
aPara[3].sParaName := 'colTinyInt';  aPara[3].eParaType := E_ExpParameterType.Byte;  aPara[3].nParaSize := 1;
aPara[4].sParaName := 'colBit';  aPara[4].eParaType := E_ExpParameterType.Boolean;  aPara[4].nParaSize := 1;
aPara[5].sParaName := 'colMoney';  aPara[5].eParaType := E_ExpParameterType.Double64;  aPara[5].nParaSize := 8;
aPara[6].sParaName := 'colFloat';  aPara[6].eParaType := E_ExpParameterType.Double64;  aPara[6].nParaSize := 8;
aPara[7].sParaName := 'colReal';  aPara[7].eParaType := E_ExpParameterType.Float32;  aPara[7].nParaSize := 4;
aPara[8].sParaName := 'colDateTime';  aPara[8].eParaType := E_ExpParameterType.DateTime;  aPara[8].nParaSize := 4;
aPara[9].sParaName := 'colNText';  aPara[9].eParaType := E_ExpParameterType.STRING;  aPara[9].nParaSize := 256;
aPara[10].sParaName := 'colNChar';  aPara[10].eParaType := E_ExpParameterType.STRING;  aPara[10].nParaSize := 10;
aPara[12].sParaName := 'colNVarChar';  aPara[12].eParaType := E_ExpParameterType.STRING;  aPara[12].nParaSize := 50;
aPara[13].sParaName := 'colBinary';  aPara[13].eParaType := E_ExpParameterType.ByteArray;  aPara[13].nParaSize := 30;
aPara[14].sParaName := 'colVarBinary';  aPara[14].eParaType := E_ExpParameterType.ByteArray;  aPara[14].nParaSize := 20;

// set command

// call functionblock
IF fbPLCDBCmd.Execute(
  hDBID:= 1,
  pExpression:= ADR(sCmd),
  cbExpression:= SIZEOF(sCmd),
  pData:= ADR(myStruct),
  cbData:= SIZEOF(myStruct),
  pParameter:= ADR(aPara),
  cbParameter:= SIZEOF(aPara))
THEN
  IF fbPLCDBCmd.bError THEN
    tcMessage := fbPLCDBCmd.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF
ExecuteDataReturn

This method can be used to send SQL commands to the database. The database connection is opened with each call and then closed again. It is possible to define placeholders in the command, which are then replaced by the TwinCAT Database Server with the corresponding values before the execution. A specified number of records can be read.

Syntax

METHOD ExecuteDataReturn : BOOL
VAR_INPUT
  hDBID: UDINT;
pExpression: POINTER TO BYTE;
cbExpression: UDINT;
pData: POINTER TO BYTE;
cbData: UDINT;
pParameter: POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter;
cbParameter: UDINT;
nStartIndex: UDINT;
nRecordCount: UDINT;
pReturnData: POINTER TO BYTE;
cbReturnData: UDINT;
pRecords: POINTER TO UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
<tr>
<td>pExpression</td>
<td>POINTER TO BYTE</td>
<td>Address of the string variable with the SQL command.</td>
</tr>
<tr>
<td>cbExpression</td>
<td>UDINT</td>
<td>Length of the string variable with the SQL command.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure with the parameter values</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Length of the structure with the parameter values</td>
</tr>
<tr>
<td>pParameter</td>
<td>POINTER TO ARRAY[0..MAX_DBCOLUMNS] OF ST_ExpParameter</td>
<td>Address of the structure array with the parameter information.</td>
</tr>
<tr>
<td>cbParameter</td>
<td>UDINT</td>
<td>Length of the structure array with the parameter information.</td>
</tr>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pReturnData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbReturnData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
<tr>
<td>pRecords</td>
<td>POINTER TO BYTE</td>
<td>Number of read records.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteDataReturn</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Parameterizing the command

The column names for the individual parameters are specified in curly brackets in the SQL command.
Sample: 'SELECT * FROM MyHouse_Temperatures WHERE Room = {SelectedRoom}'.
Accordingly, SelectedRoom has to be specified as parameter name in the structure ST_ExpParameter.

Some databases do not support the parameterization of SQL clauses. (TOP/LIMIT/ROWNUM/...)
Parameterizable table names are not usually supported.
VAR
  fbPLCDBCmd : FB_PLCDBCmdEvt(sNetID := '', tTimeout := T#5S);
  sCmd : STRING (1000);
  stPara : ST_ExpParameter;
  RecordAmt : UUINT := 3;
  ReturnDataStruct : ARRAY [0..9] OF ST_DataAll;
  nRecords : UDINT;
  tcMessage : I_TcMessage;
END_VAR

// set Parameter configuration
stPara.eParaType := E_ExpParameterType.Int64;
stPara.nParaSize := 8;
stPara.sParaName := 'RecordAmt';

// set command with placeholder
sCmd := 'SELECT TOP ({RecordAmt}) * FROM MyTableName';

// call functionblock
IF fbPLCDBCmd.ExecuteDataReturn(
  hDBID:= 1,
  pExpression:= ADR(sCmd),
  cbExpression:= SIZEOF(sCmd),
  pData:= ADR(RecordAmt),
  cbData:= SIZEOF(RecordAmt),
  pParameter:= ADR(stPara),
  cbParameter:= SIZEOF(stPara),
  nStartIndex:= 0,
  nRecordCount:= 10,
  pReturnData:= ADR(ReturnDataStruct),
  cbReturnData:= SIZEOF(ReturnDataStruct),
  pRecords:= ADR(nRecords))
THEN
  IF fbPLCDBCmd.bError THEN
    tcMessage := fbPLCDBCmd.ipTcResult;
    nState := 255;
  ELSE
    nState := 0;
  END_IF
END_IF
6.1.4.3 SQL Expert mode

6.1.4.3.1 FB_ConfigTcDBSvc

Function block for creating, reading and deleting configuration entries for the TwinCAT Database Server.

Syntax

Definition:

```
FUNCTION_BLOCK FB_ConfigTcDBSvc
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent;
END_VAR
```
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create [153]</td>
<td>Local</td>
<td>Creates new entries in the XML configuration file for the TwinCAT Database Server</td>
</tr>
<tr>
<td>Read [154]</td>
<td>Local</td>
<td>Reads the current configuration of the TwinCAT Database Server</td>
</tr>
<tr>
<td>Delete [155]</td>
<td>Local</td>
<td>Deletes the database and AutoLog groups from the configuration of the TwinCAT Database Server</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Create**

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

**Syntax**

METHOD Create : BOOL
VAR_INPUT
  pTcDBSrvConfig: POINTER TO BYTE;
  cbTcDBSrvConfig: UDINT;
  bTemporary: BOOL := TRUE;
  pConfigID: POINTER TO UDINT;
END_VAR
Creating AutoLog groups is currently not supported.

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

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

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

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

Read

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

Syntax

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDBConfig</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigDB</td>
<td>Pointer address of the array into which the database configurations are to be written.</td>
</tr>
<tr>
<td>cbDBConfig</td>
<td>UDINT</td>
<td>Length of the database configuration array</td>
</tr>
<tr>
<td>pAutoLogGrpConfig</td>
<td>POINTER TO ARRAY [1..MAX_CONFIGURATIONS] OF ST_ConfigAutoLogGrp</td>
<td>Pointer address of the array into which the AutoLogGrp configurations are to be written.</td>
</tr>
<tr>
<td>cbAutoLogGrpConfig</td>
<td>UDINT</td>
<td>Length of the AutoLogGrp configuration array</td>
</tr>
<tr>
<td>pDBCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of database configurations.</td>
</tr>
<tr>
<td>pAutoLogGrpCount</td>
<td>POINTER TO UDINT</td>
<td>Pointer address for storing the number of AutoLogGrp configurations.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

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

**Delete**

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

**Syntax**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTcDBSrvConfigType</td>
<td>E_TcDBSrvConfigType</td>
<td>Type of the configuration to be deleted (database / AutoLog group)</td>
</tr>
<tr>
<td>hConfigID</td>
<td>UDINT</td>
<td>ID of the configuration to be deleted (hDBID or hAutoLogGrpID)</td>
</tr>
</tbody>
</table>

## Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

## Sample

```plaintext
VAR
    fbConfigTcDBSrv : FB_ConfigTcDBSrv(sNetId := '', tTimeout:=T#5S);
    myConfigHandle : INT;
END_VAR

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

## 6.1.4.3.2 FB_SQLDatabase

**Function block for opening, closing and managing a database connection.**

### Syntax

**Definition:**

```plaintext
FUNCTION BLOCK FB_SQLDatabase
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>
**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect [279]</td>
<td>Local</td>
<td>Opens a connection to a declared database.</td>
</tr>
<tr>
<td>CreateCmd [280]</td>
<td>Local</td>
<td>Initializes an instance of the function block FB_SQLCommand with the already open database connection of the function block FB_SQLDatabase.</td>
</tr>
<tr>
<td>CreateSP [280]</td>
<td>Local</td>
<td>Initializes an instance of the function block FB_SQLStoredProcedure with the already open database connection of the function block FB_SQLDatabase.</td>
</tr>
<tr>
<td>Disconnect [281]</td>
<td>Local</td>
<td>Closes the connection to the database that was opened by this function block instance.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Connect**

This method opens a connection to a declared database.

**Syntax**

```
METHOD Connect : BOOL
VAR_INPUT
  hDBID: UDINT := 1;
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
<td>Indicates the ID of the database to be used.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

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

// open connection
IF fbSqlDatabase.Connect(1) THEN
  IF fbSqlDatabase.bError THEN
```

---

TF6420  Version: 1.11  279
CreateCmd

This method is used to initialize an instance of the function block FB_SQLCommand with the already open database connection of the function block FB_SQLDatabase. The function block FB_SQLCommand only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLCommand can be initialized with the same database connection.

The initialization of the function block FB_SQLCommand is completed in the same cycle. This means that neither the Busy flag of the function block nor the method return value of the CreateCmd method have to be checked.

Syntax

METHOD CreateCmd : BOOL
VAR_INPUT
   pSQLCommand: POINTER TO FB_SQLCommand;
END_VAR

_inputs_

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSQLCommand</td>
<td>POINTER TO FB_SQLCommand</td>
<td>Returns a new instance of the function block FB_SQLCommand.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCmd</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

VAR
   fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR

// create a command reference
IF fbSqlDatabase.CreateCmd(ADR(fbSqlCommand)) THEN
   IF fbSqlDatabase.bError THEN
      nState := 255;
   ELSE
      nState := nState+1;
   END_IF
END_IF

FB_SQLCommandEvt [192] can then be used for the execution.

CreateSP

This method is used to initialize an instance of the function block FB_SQLStoredProcedure with the already open database connection of the function block FB_SQLDatabase. The function block FB_SQLStoredProcedure only uses the database connection it was assigned via the CreateCmd method. Several instances of the function block FB_SQLStoredProcedure can be initialized with the same database connection.

The initialization of the function block FB_SQLStoredProcedure may take several cycles. The Busy flag of the function block or the method return value of the CreateCmd method have to be checked before the function block can be used.
### Syntax

METHOD CreateSP : BOOL
VAR_INPUT
  sProcedureName: T_MaxString;
  pParameterInfo: POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter;
  cbParameterInfo: UDINT;
  pSQLProcedure: POINTER TO FB_SQLStoredProcedure;
END_VAR

#### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sProcedureName</td>
<td>T_MaxString</td>
<td>Indicates the name of the procedure to be executed.</td>
</tr>
<tr>
<td>pParameterInfo</td>
<td>POINTER TO ARRAY [0..MAX_SPPARAMETER] OF ST_SQLSPParameter</td>
<td>Pointer address for the parameter info list.</td>
</tr>
<tr>
<td>cbParameterInfo</td>
<td>UDINT</td>
<td>Indicates the length of the parameter info list.</td>
</tr>
<tr>
<td>pSQLProcedure</td>
<td>POINTER TO FB_SQLStoredProcedure</td>
<td>Returns a new instance of the function block FB_SQLStoredProcedure.</td>
</tr>
</tbody>
</table>

#### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateSP</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

#### Sample

```plaintext
VAR
  fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
  ParaInfo : ST_SQLSPParameter;
END_VAR
ParaInfo.sParameterName := '@Customer_ID';
ParaInfo.eParameterType := E_SPParameterType.Input;
ParaInfo.eParameterDataType := E_ColumnType.BigInt;
ParaInfo.nParameterSize := 8;
IF fbSQLDatabase.CreateSP('dbo.SP_GetCustomerPositions', ADR(ParaInfo), SIZEOF(ParaInfo), ADR(fbSQLStoredProcedure)) THEN
  IF fbSQLDatabase.bError THEN
    nState:=255;
  ELSE
    nState:= nState+1;
  END_IF
END_IF
```

Subsequently, the FB_SQLStoredProcedureEvt can be used to execute the stored procedure.

### Disconnect

This method closes the connection to the database that was opened by this function block instance.

#### Syntax

METHOD Disconnect : BOOL

#### Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disconnect</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>
Sample

VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
END_VAR

// disconnect from database
IF fbSqlDatabase.Disconnect() THEN
    IF fbSqlDatabase.bError THEN
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF

6.1.4.3.3 FB_SQLCommand

Function block for executing SQL commands. Before it can be used it has to be initialized with the function block FB_SQLDatabase.

Syntax

Definition:

FUNCTION BLOCK FB_SQLCommand
VAR_INPUT
    sNetID: T_AmsNetID := '';
    tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
    bBusy: BOOL;
    bError: BOOL;
    ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute [283]</td>
<td>Local</td>
<td>Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase.</td>
</tr>
<tr>
<td>ExecuteDataReturn [284]</td>
<td>Local</td>
<td>Sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the function block FB_SQLResult can be transferred for reading the returned records.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Execute

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase.

Syntax

METHOD Execute : BOOL
VAR_INPUT
  pSQLCmd: POINTER TO BYTE;
  cbSQLCmd: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSQLCmd</td>
<td>POINTER TO BYTE</td>
<td>Indicates the pointer address of a string variable with the SQL command to be executed.</td>
</tr>
<tr>
<td>cbSQLCmd</td>
<td>UDINT</td>
<td>Indicates the length of a SQL command to be executed.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>POINTER TO BYTE</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

Uses the command created by FB_SQLDatabaseEvt.CreateCmd() [188].

VAR
  fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
  tcMessage    : I_TcMessage;
END_VAR

// you can generate this with the SQL Query Editor
sCmd := 'INSERT INTO myTable_Double (Timestamp, Name, Value) VALUES ( $'2018-01-31 14:59:27$', $'Temperature$', 21.3)';

// call sql command
IF fbSQLCommand.Execute(ADR(sCmd), SIZEOF(sCmd)) THEN
  IF fbSQLCommand.bError THEN
    tcMessage := fbSQLCommand.ipTcResult;
  ELSE
    // process the result
  END_IF
END_IF
nState := 255;
ELSE
  nState := nState + 1;
END_IF
END_IF

ExecuteDataReturn

This method sends the specified SQL command to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the function block FB_SQLResult can be transferred for reading the returned records.

Syntax

METHOD ExecuteDataReturn : BOOL
VAR_INPUT
  pSQLCmd: POINTER TO BYTE;
  cbSQLCmd: UDINT;
  pSQLDBResult: POINTER TO FB_SQLResult;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSQLCmd</td>
<td>POINTER TO BYTE</td>
<td>Indicates the pointer address of a string variable with the SQL command to be executed.</td>
</tr>
<tr>
<td>cbSQLCmd</td>
<td>UDINT</td>
<td>Indicates the length of a SQL command to be executed.</td>
</tr>
<tr>
<td>pSQLDBResult</td>
<td>POINTER TO FB_SQLResult</td>
<td>Returns a new instance of the function block FB_SQLResult.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteDataReturn</td>
<td>POINTER TO BYTE</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

Uses the command created by FB_SQLDatabaseEvt.CreateCmd() [188].

VAR
  fbSqlCommand : FB_SQLCommandEvt(sNetID := '', tTimeout := T#5S);
  tcMessage    : I_TcMessage;
END_VAR

// you can generate this with the SQL Query Editor
sCmd := 'SELECT ID, Timestamp, Name, Value FROM myTable_Double';

// call sql command
IF fbSQLCommand.ExecuteDataReturn(ADR(sCmd), SIZEOF(sCmd), ADR(fbSqlResult)) THEN
  IF fbSQLCommand.bError THEN
    nState := 255;
  ELSE
    tcMessage := fbSQLCommand.ipTcResult;
    nState := nState + 1;
  END_IF
END_IF

FB_SQLResultEvt [195] can then be used to read the data.
The function block is used for reading the cached records.

**Syntax**

**Definition:**

```plaintext
FUNCTION BLOCK FB_SQLResult
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Local</td>
<td>Reads a specified number of records from the result data cached in the TwinCAT Database Server.</td>
</tr>
<tr>
<td>Release</td>
<td>Local</td>
<td>Releases data buffered by the TwinCAT Database Server.</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

**Read**

This method reads a specified number of records from the result data cached in the TwinCAT Database Server.
**Syntax**

METHOD Read : BOOL
VAR_INPUT
  nStartIndex: UDINT := 0;
  nRecordCount: UDINT := 1;
  pData: POINTER TO BYTE;
  cbData: UDINT;
  bWithVerifying: BOOL := FALSE;
  bDataRelease: BOOL := TRUE;
END_VAR

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStartIndex</td>
<td>UDINT</td>
<td>Indicates the index of the first record to be read.</td>
</tr>
<tr>
<td>nRecordCount</td>
<td>UDINT</td>
<td>Indicates the number of records to be read.</td>
</tr>
<tr>
<td>pData</td>
<td>POINTER TO BYTE</td>
<td>Address of the structure array into which the records are to be written.</td>
</tr>
<tr>
<td>cbData</td>
<td>UDINT</td>
<td>Indicates the size of the structure array in bytes.</td>
</tr>
<tr>
<td>bWithVerifying</td>
<td>BOOL</td>
<td>Return data are compared with the pData structure array and adjusted if necessary.</td>
</tr>
<tr>
<td>bDataRelease</td>
<td>BOOL</td>
<td>Releases the cached data.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

**Sample**

VAR
  fbSqlResult : FB_SQLResultEvt(sNetID:='', tTimeout := T#5S);
  aReadStruct : ARRAY[1..5] OF ST_StandardRecord;
END_VAR

// get values from internal tc db srv storage
IF fbSqlResult.Read(2, 3, ADR(aReadStruct), SIZEOF(aReadStruct), FALSE, TRUE) THEN
  IF fbSqlResult.bError THEN
    nState := 255;
  ELSE
    nState := nState+1;
  END_IF
ENDIF

**Result in the PLC:**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aReadStruct[1]</td>
<td>ST_StandardRecord</td>
<td></td>
</tr>
<tr>
<td>nID</td>
<td>LINT</td>
<td>9</td>
</tr>
<tr>
<td>dtTimestamp</td>
<td>DATE_AND_TIME</td>
<td>DT#2018-1-31-15:45:59</td>
</tr>
<tr>
<td>sName</td>
<td>STRING(80)</td>
<td>'Temperature'</td>
</tr>
<tr>
<td>rValue</td>
<td>LREAL</td>
<td>21.3</td>
</tr>
<tr>
<td>aReadStruct[2]</td>
<td>ST_StandardRecord</td>
<td></td>
</tr>
<tr>
<td>nID</td>
<td>LINT</td>
<td>10</td>
</tr>
<tr>
<td>dtTimestamp</td>
<td>DATE_AND_TIME</td>
<td>DT#2018-1-31-15:5:59</td>
</tr>
<tr>
<td>sName</td>
<td>STRING(80)</td>
<td>'Temperature'</td>
</tr>
<tr>
<td>rValue</td>
<td>LREAL</td>
<td>21.2</td>
</tr>
</tbody>
</table>
Release

This method can be used to release data cached by the TwinCAT Database Server.

Syntax

METHOD Release : BOOL

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

6.1.4.3.5 FB_SQLStoredProcedure

Function block for executing stored procedures of the database. Before it can be used it has to be initialized with the function block "FB_SQLDatabase".

Syntax

Definition:

FUNCTION_BLOCK FB_SQLStoredProcedure
VAR_INPUT
  sNetID: T_AmsNetID := '';
  tTimeout: TIME := T#5S;
END_VAR
VAR_OUTPUT
  bBusy: BOOL;
  bError: BOOL;
  ipTcResultEvent: Tc3_EventLogger.I_TcResultEvent
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID</td>
<td>T_AmsNetID</td>
<td>AMS network ID of the target device at which the ADS command is directed.</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
<td>Indicates the time before the function is cancelled.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy</td>
<td>BOOL</td>
<td>TRUE as soon as a method of the function block is active.</td>
</tr>
<tr>
<td>bError</td>
<td>BOOL</td>
<td>TRUE when an error occurs.</td>
</tr>
<tr>
<td>ipTcResultEvent</td>
<td>Tc3_EventLogger.I_TcResultEvent</td>
<td>Result interface with detailed information on the return value.</td>
</tr>
</tbody>
</table>
Methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute [288]</td>
<td>Local</td>
<td>Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase.</td>
</tr>
<tr>
<td>ExecuteDataReturn [289]</td>
<td>Local</td>
<td>Sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the FB_SQLResult function block can be transferred for reading the returned records.</td>
</tr>
<tr>
<td>Release [289]</td>
<td>Local</td>
<td>Releases the parameter information of the stored procedure that was transferred during initialization.</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.1 Build 4020.10</td>
<td>PC or CX (x86)</td>
<td>Tc3_Database</td>
</tr>
</tbody>
</table>

Execute

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase.

Syntax

METHOD Execute : BOOL
VAR_INPUT
   pParameterStrc: POINTER TO BYTE;
   cbParameterStrc: UDINT;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pParameterStrc</td>
<td>POINTER TO BYTE</td>
<td>Pointer address to the parameter structure that is transferred to the procedure.</td>
</tr>
<tr>
<td>cbParameterStrc</td>
<td>UDINT</td>
<td>Length of the parameter structure</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [188].

VAR
   fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID:='', tTimeout := T#5S);
   Customer_ID : LINT;
   tcMessage : I_TcMessage;
END_VAR

IF fbSQLStoredProcedure.Execute(pParameterStrc := ADR(Customer_ID) , cbParameterStrc:= SIZEOF(Customer_ID)) THEN
   IF fbSQLStoredProcedure.bError THEN
     tcMessage := fbSQLStoredProcedure.ipTcResult;
     nState := 255;
   ELSE
     IF fbSQLStoredProcedure.bSuccess THEN
       nState := 0;
     ELSE
       nState := 255;
     END_IF
   END_IF
END_IF

288 Version: 1.11 TF6420
ExecuteDataReturn

This method sends the call of the specified stored procedure to the database via the database connection already opened by the function block FB_SQLDatabase. An instance of the FB_SQLResult function block can be transferred for reading the returned records.

Syntax

METHOD ExecuteDataReturn : BOOL
VAR_INPUT
    pParameterStrc: POINTER TO BYTE;
    cbParameterStrc: UDINT;
    pSQLDBResult: POINTER TO FB_SQLDBResult;
END_VAR

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pParameterStrc</td>
<td>POINTER TO BYTE</td>
<td>Pointer address to the parameter structure that is transferred to the procedure.</td>
</tr>
<tr>
<td>cbParameterStrc</td>
<td>UDINT</td>
<td>Length of the parameter structure</td>
</tr>
<tr>
<td>pSQLDBResult</td>
<td>POINTER TO FB_SQLDBResult</td>
<td>Returns a new instance of the function block FB_SQLDBResult.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

Sample

Uses the stored procedure previously created with FB_SQLDatabaseEvt.CreateSP() [188].

VAR
    fbSQLStoredProcedure : FB_SQLStoredProcedureEvt(sNetID:='', tTimeout := T#5S);
    Customer_ID : LINT;
    tcMessage    : I_TcMessage;
END_VAR

IF fbSQLStoredProcedure.ExecuteDataReturn(pParameterStrc := ADR(Customer_ID), cbParameterStrc:= SIZEOF(Customer_ID), pSQLDBResult := ADR(fbSqlResult)) THEN
    IF fbSQLStoredProcedure.bError THEN
        tcMessage := fbSQLStoredProcedure.ipTcResult;
        nState := 255;
    ELSE
        nState := nState+1;
    END_IF
END_IF

FB_SQLResultEvt [195] can then be used to read the data.

Release

This method releases the parameter information of the stored procedure, which was transferred during initialization.
Syntax

METHOD Release : BOOL

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>BOOL</td>
<td>Displays the status of the method. Returns TRUE as soon as the method execution is finished, even in the event of an error.</td>
</tr>
</tbody>
</table>

6.2 Tc2_Database

Overview

The Tc2_Database library contains function blocks for controlling and configuring the TwinCAT 3 database server.
### Function blocks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_DBReloadConfig [p. 293]</td>
<td>Reloads the XML configuration file</td>
</tr>
<tr>
<td>FB_GetAdsDevXMLConfig [p. 298]</td>
<td>Reads all ADS device configurations from the XML configuration file.</td>
</tr>
<tr>
<td>FB_DBConnectionOpen [p. 299]</td>
<td>Opens a connection to a database.</td>
</tr>
<tr>
<td>FB_DBConnectionClose [p. 300]</td>
<td>Closes a connection to a database.</td>
</tr>
<tr>
<td>FB_DBCreate [p. 301]</td>
<td>Creates a new database</td>
</tr>
<tr>
<td>FB_DBTableCreate [p. 302]</td>
<td>Creates a table with any desired table structure</td>
</tr>
<tr>
<td>FB_DBRead [p. 304]</td>
<td>Reads one value out of the database</td>
</tr>
<tr>
<td>FB_DBWrite [p. 305]</td>
<td>Writes one variable value, with timestamp, into a database</td>
</tr>
<tr>
<td>FB_DBCyclicRdWrt [p. 303]</td>
<td>Starts or stops the logging/writing of variables</td>
</tr>
<tr>
<td>FB_DBRecordSelect [p. 315]</td>
<td>Reads a data record out of a table</td>
</tr>
<tr>
<td>FB_DBRecordSelect_EX [p. 317]</td>
<td>Reads a data record out of a table (command length less than 10,000 characters)</td>
</tr>
<tr>
<td>FB_DBRecordArraySelect [p. 308]</td>
<td>Reads several records from a table.</td>
</tr>
<tr>
<td>FB_DBRecordInsert [p. 314]</td>
<td>Creates a new data record</td>
</tr>
<tr>
<td>FB_DBRecordInsert_EX [p. 307]</td>
<td>Creates a new data record. (command length less than 10,000 characters)</td>
</tr>
<tr>
<td>FB_DBRecordDelete [p. 306]</td>
<td>Deletes a record from a table</td>
</tr>
<tr>
<td>FB_DBStoredProcedures [p. 311]</td>
<td>Executes a stored procedure</td>
</tr>
<tr>
<td>FB_DBStoredProceduresRecordReturn [p. 318]</td>
<td>Executes a stored procedure and returns a record</td>
</tr>
<tr>
<td>FB_DBStoredProceduresRecordArray [p. 312]</td>
<td>Executes a stored procedure and returns several records.</td>
</tr>
</tbody>
</table>
Data Types

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_DBColumnCfg</td>
</tr>
<tr>
<td>ST_DBXMLCfg</td>
</tr>
<tr>
<td>ST_ADSDevXMLCfg</td>
</tr>
<tr>
<td>ST_DBSQLError</td>
</tr>
<tr>
<td>ST_DBParameter</td>
</tr>
<tr>
<td>E_DbColumnTypes</td>
</tr>
<tr>
<td>E_DBTypes</td>
</tr>
<tr>
<td>E_DbValueType</td>
</tr>
<tr>
<td>E_DBWriteModes</td>
</tr>
<tr>
<td>E_DBParameterTypes</td>
</tr>
</tbody>
</table>

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1 Function blocks

6.2.1.1 FB_GetStateTcDatabase

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

VAR_OUTPUT
   bBusy     : BOOL;
   bError    : BOOL;
   nErrID    : UDINT;
   nAdsState : UINT;
   nDevState : UINT;
END_VAR
```

The function block allows to get the current state of the Twincat Database Server.

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

bExecute: The command is executed with the rising edge.

tTimeout: Indicates the timeout time.
bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code if the bError output is set.

nAdsState: Contains the state identification code of the ADS target device. The codes returned here are specified for all ADS servers:

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

nDevState: Contains the specific state identification code of the ADS target device. The codes returned here are supplementary information specific to the ADS device.

- 1 = TwinCAT Database Server started
- 2 = cyclic reading or writing started

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1.2   FB_DBReloadConfig

With the FB_DBReloadConfig function block the XML configuration file can be reloaded. If the XML configuration file was modified, the Database Server must be notified of the modifications with the aid of FB_DBReloadConfig.

VAR_INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID             : T_AmsNetId;</td>
<td></td>
</tr>
<tr>
<td>bExecute           : BOOL;</td>
<td></td>
</tr>
<tr>
<td>tTimeout           : TIME;</td>
<td></td>
</tr>
</tbody>
</table>

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

bExecute: The command is executed with a rising edge.

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.
VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code if the bError output is set.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1.3 FB_DBConnectionAdd

The FB_DBConnectionAdd function block permits additional database connections to be added to the XML configuration file.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

eDBType: Indicates the type of the database, e.g. 'Mobile server'.

eDBValueType: Indicates the form in which the values are or will be stored.

sDBServer: Provides the name of the server: Optional.

sDBProvider: Gives the provider of the database: Optional.

sDBUrl: Gives the path to the database.
sSystemDB: Only for Access databases. Indicates the path to the MDW file.

sUserId: Indicates the login user name.

sPassword: Indicates the password.

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

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR OUTPUT

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy: BOOL;</td>
</tr>
<tr>
<td>bError: BOOL;</td>
</tr>
<tr>
<td>bErrID: UDINT;</td>
</tr>
<tr>
<td>hDBID: UDINT;</td>
</tr>
</tbody>
</table>

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code if the bError output is set.

hDBID: Returns the ID of the database.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1.4 FB_DBOdbcConnectionAdd

The function block FB_DBOdbcConnectionAdd can be used to add further ODBC database connections to the XML configuration file.

VAR INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID: T_AmsNetId;</td>
</tr>
<tr>
<td>eDBType: E_DBTypes;</td>
</tr>
<tr>
<td>eDBValueType: E_DBValueType;</td>
</tr>
<tr>
<td>sDBDriver: T_MaxString;</td>
</tr>
</tbody>
</table>

The function block FB_DBOdbcConnectionAdd can be used to add further ODBC database connections to the XML configuration file.
sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

eDBType: Indicates the type of the database, e.g. ‘Mobile server’.

eDBValueType: Indicates the form in which the values are or will be stored.

sDBDriver: Indicates the name of the ODBC driver to be used.

sDBServer: Indicates the name of the server.

sDBDatabase: Indicates the name of the database.

nDBPort: Indicates the port for the ODBC connection.

sDBProtocol: Indicates the protocol to be used (TCPIP).

sDBUserId: Indicates the user name.

sDBPassword: Indicates the password to be used.

sDBScheme: Indicates the database schema to be used.

sDBSequence: Indicates the sequence name for Oracle databases.

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

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

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code if the bError output is set.

hDBID: Returns the ID of the database.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
6.2.1.5  FB_AdsDeviceConnectionAdd

The function block FB_AdsDeviceConnectionAdd permits additional Ads-Device connections to be added to the XML configuration file.

**VAR_INPUT**

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

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
- **sADSDevNetID**: String containing the AMS network ID of the ADS device.
- **nADSDevPort**: Indicates the port of the ADS device.
- **tADSDevTimeout**: Indicates the timeout time of the ADS device.
- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: Indicates the duration of the timeout.

**VAR_OUTPUT**

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

- **bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
- **bError**: Becomes TRUE, as soon as an error occurs.
- **nErrID**: Returns the ADS error code if the bError output is set.
- **hAdsId**: Returns the ID of the ADS device.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
6.2.1.6 FB_GetDBXMLConfig

With this function block FB_GetDBXMLConfig all declared databases can be read out of the XML-configuration file.

VAR_INPUT

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

VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrID : UDINT;

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

cbDBCfg: Indicates the length of the array, into which the configurations are to be written.

pDBCfg: Indicates the pointer address of the array, into which the configurations are to be written.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code if the bError output is set.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1.7 FB_GetAdsDevXMLConfig

VAR_INPUT

sNetID : T_AmsNetId;
cbAdDevCfg : UDINT;
pAdDevCfg : POINTER TO ARRAY [0.. MAX_XML_DECLARATIONS] OF ST_AdvDevXMLCfg
bExecute : BOOL;
tTimeout : TIME;

VAR_OUTPUT

bBusy : BOOL;
bError : BOOL;
nErrID : UDINT;

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code if the bError output is set.
With this function block FB_GetAdsDevXMLConfig all declared ADS-devices can be read out of the XML-configuration file.

**VAR_INPUT**

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

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
- **cbAdsDevCfg**: Indicates the length of the array, into which the configurations are to be written.
- **pAdsDevCfg**: Indicates the pointer address of the array, into which the configurations are to be written.
- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

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

- **bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
- **bError**: Becomes TRUE, as soon as an error occurs.
- **nErrID**: Returns the ADS error code if the bError output is set.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

### 6.2.1.8 FB_DBConnectionOpen

You can open connections to databases with this function block FB_DBConnectionOpen. This can improve the read and write access speed with the function blocks FB_DBWrite, FB_DBRead, FB_DBRecordInsert and FB_FBRecordSelect.

**VAR_INPUT**

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

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
hDBID: Indicates the ID of the database to be used.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>nErrID : UDINT;</td>
</tr>
<tr>
<td>sSQLState : ST_DBSQLError;</td>
</tr>
</tbody>
</table>

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

**6.2.1.9 FB_DBConnectionClose**

The function block FB_DBConnectionClose can be used to make connections with databases. If a connection with a database was opened previously, it must be closed again.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID : T_AmsNetId;</td>
</tr>
<tr>
<td>hDBID : DINT;</td>
</tr>
<tr>
<td>bExecute : BOOL;</td>
</tr>
<tr>
<td>tTimeout : TIME;</td>
</tr>
</tbody>
</table>

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>bBusy : BOOL;</td>
</tr>
<tr>
<td>bError : BOOL;</td>
</tr>
<tr>
<td>nErrID : UDINT;</td>
</tr>
</tbody>
</table>

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.

sSQLState: Returns the SQL error code [320] of the corresponding database type.
**bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

**nErrID**: Returns the ADS error code if the bError output is set.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

### 6.2.1.10 FB_DBCreate

The FB_DBCreate function block allows databases to be created.

The following database types can be created with this function block: MS SQL databases, MS SQL Compact databases, MS Access databases and XML databases.

ASCII files can (but do not have to) be created with the function block FB_DBCreate. If they do not exist, they are created automatically during the first write access. They only have to be declared in the XML configuration file.

It is not possible to create DB2, Oracle, MySQL, PostgreSQL, InterBase and Firebird databases. In addition, it is not possible to overwrite existing databases. In this case the function block FB_DBCreate would return an error.

#### VAR_INPUT

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

**sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.

**sPathName**: Gives the path to the database.

**sDBName**: Gives the name of the database that is to be created.

**eDBType**: Gives the type of the database that is to be created.

**sSystemDB**: Only for Access databases. Contains the path to the MDW file.

**sUserID**: User name for the corresponding registration

**sPassword**: Corresponding password
**bExecute**: The command is executed with a rising edge.

**tTimeout**: Indicates the duration of the timeout.

### VAR_OUTPUT

<table>
<thead>
<tr>
<th>bBusy</th>
<th>BOOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>bError</td>
<td>BOOL</td>
</tr>
<tr>
<td>nErrID</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

**bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

**nErrID**: Returns the ADS error code if the bError output is set.

---

**TwinCAT Database Server**

If the newly created databases are to be used by the TwinCAT Database Server, the connection data have to be written to the XML configuration file with the aid of the function block FB_DBConnectionADD.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

---

#### 6.2.1.11 FB_DBTableCreate

The FB_DBTableCreate function block permits tables with any desired table structure to be created in databases.

### VAR_INPUT

<table>
<thead>
<tr>
<th>sNetID</th>
<th>T_AmsNetID</th>
</tr>
</thead>
<tbody>
<tr>
<td>hDBID</td>
<td>UDINT</td>
</tr>
<tr>
<td>sTableName</td>
<td>T_MaxString</td>
</tr>
<tr>
<td>cbTableCfg</td>
<td>UDINT</td>
</tr>
<tr>
<td>pTableCfg</td>
<td>POINTER TO ARRAY[0..MAX_DB_TABLE_COLUMNS] OF ST_DBColumnCfg</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
</tr>
<tr>
<td>tTimeout</td>
<td>TIME</td>
</tr>
</tbody>
</table>

**sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

**sTableName**: Provides the name of the table.

**cbTableCfg**: Returns the length of the array in which the columns are configured.
pTableCfg: Provides the pointer address of the table structure array. The individual columns are written in this array.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the duration of the timeout.

**VAR_OUTPUT**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

**6.2.1.12 FB_DBCyclicRdWrt**

The FB_DBCyclicRdWrt function block can be used to start or stop the cyclic logging \ writing of variables.

**VAR_INPUT**

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

bExecute: The read/write cycle is started with a rising edge and stopped with a falling edge.

tTimeout: States the length of the timeout that may not be exceeded by execution of the ADS command.

**VAR_OUTPUT**

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
**bError**: Becomes TRUE, as soon as an error occurs.

**nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.

**sSQLState**: Returns the SQL error code [320] of the corresponding database type.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

#### 6.2.1.13 FB_DBRead

The FB_DBRead allows values to be read from a database.

**VAR_INPUT**

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

**sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

**sDBVarName**: Gives the name of the variable that is to be read.

**cbReadLen**: Indicates the length of the buffer that is to be read.

**pDestAddr**: Contains the address of the buffer which is to receive the data that has been read.

**bExecute**: The command is executed with a rising edge.

**tTimeout**: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

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

**bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

**nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.
**sSQLState:** Returns the SQL error code [320] of the corresponding database type.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

### 6.2.1.14 FB_DBWrite

The **FB_DBWrite** function block can be used to write the values of individual variables into databases. The table structure must contain the columns "Timestamp", "Name", and "Value" (see “SQL Compact database” [126]). In order to be able to use the function block, the database that is to be used for write access and the ADS device, from which the variables are to be read, must be declared in the XML configuration file.

**VAR_INPUT**

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

- **sNetID:** String containing the AMS network ID of the target device, at which the ADS command is directed.
- **hDBID:** ID of the database to be used.
- **hAdsID:** ID of the ADS device to be used.
- **sVarName:** Provides the name of the variable.
- **nIGroup:** Index group of the variable (optional, only on the BC9000).
- **nIOffset:** Index offset of the variable (optional, only on the BC9000).
- **nVarSize:** Size of the variable in bytes (optional, only on the BC9000).
**sVarType**: Data type of the variable (optional, only on the BC9000).

Possible variable data types: "BOOL" / "LREAL" / "REAL" / "INT16" / "DINT" / "USINT" / "BYTE" / "UDINT" / "DWORD" / "UINTEGER" / "WORD" / "SINT"

**sDBVarName**: Variable name to be used in the database.

**eDBWriteMode**: Indicates whether the values are to be appended in new records or whether the existing records are to be updated.

**tRingBufferTime**: Indicates the maximum age of records in a table (only for Ringbuffer_WriteMode).

**nRingBufferCount**: Indicates the maximum number of records in a table (only for Ringbuffer_WriteMode).

**bExecute**: The command is executed with a rising edge.

**tTimeout**: Indicates the time before the function is cancelled.

---

**VAR_OUTPUT**

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

**bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

**nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.

**sSQLState**: Returns the SQL error code [320] of the corresponding database type.

---

**Log values of ADS devices (except 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);
```

---

**Log values of BC9000**

```
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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

---

**6.2.1.15 FB_DBRecordDelete**

```plaintext
FB_DBRECORDDELETE
  sNetID : T_AmsNetID
  bBusy : BOOL
  hDBID : UDINT
  bError : BOOL
  cbCmdSize : UDINT
  nErrID : UDINT
  pCmdAddr : UDINT
  sSQLState : ST_DBSQLError
  bExecute : BOOL
  tTimeout : TIME
```
The function block FB_DBRecordDelete can be used to delete individual records from a database. This function block can be used to execute SQL DELETE commands with up to 10,000 characters.

To use the function block it is necessary to declare the database from which the records are to be deleted in the XML configuration file.

**VAR_INPUT**

```plaintext
sNetID : T_AmsNetId;
hDBID : UDINT;
cbCmdSize: UDINT;
pCmdAddr : POINTER TO BYTE;
bExecute : BOOL;
tTimeout : TIME;
```

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
- **hDBID**: Indicates the ID of the database to be used.
- **cbCmdSize**: Indicates the length of the INSERT command.
- **pCmdAddr**: Pointer to the executing INSERT command.
- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

```plaintext
bBusy : BOOL;
bError : BOOL;
nErrID : UDINT;
sSQLState: ST_DBSQLError;
```

- **bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
- **bError**: Becomes TRUE, as soon as an error occurs.
- **nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.
- **sSQLState**: Returns the SQL error code [320] of the corresponding database type

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

**6.2.1.16 FB_DBRecordInsert_EX**

The function block FB_DBRecordInsert_EX can be used to write individual records with any structure into a database. This function block can be used to execute SQL INSERT commands with up to 10,000 characters. To use the function block it is necessary to declare the database to which the records are to be written in the XML configuration file.
VAR_INPUT

sNetID   : T_AmsNetId;
hDBID    : UDINT;
cbCmdSize: UDINT;
pCmdAddr : POINTER TO BYTE;
bExecute : BOOL;
tTimeout : TIME;
END_VAR

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

cbCmdSize: Indicates the length of the INSERT command.

pCmdAddr: Pointer to the executing INSERT command.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.

sSQLState: Returns the SQL error code [320] of the corresponding database type.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1.17 FB_DBRecordArraySelect

The function block FB_DBRecordArraySelect can be used to read several records with any structure from the database. This function block can be used to execute an SQL SELECT command with up to 10,000 characters.

This function block is not compatible with ASCII files.
VAR_INPUT

VAR_INPUT
    sNetID : T_AmsNetID;
    hDBID : UDINT;
    cbCmdSize : UDINT;
    pCmdAddr : UDINT;
    nStartIndex : UDINT;
    nRecordCount : UDINT;
    cbRecordArraySize : UDINT;
    pDestAddr : POINTER TO BYTE;
    bExecute : BOOL;
    tTimeout : TIME;
END_VAR

VAR_INPUT

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

cbCmdSize: Indicates the length of a SELECT command to be executed.

pCmdSize: Indicates the pointer address of a string variable with the SQL command to be executed.

nStartIndex: Indicates the index of the first record to be read.

nRecordCount: Indicates the number of records to be read.

cbRecordArraySize: Indicates the size of the structure array in bytes.

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

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

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

VAR_OUTPUT

ST_DBSQLError [320]

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.

sSQLState: Returns the SQL error code of the corresponding database type

nRecords: Returns the number of data records.

Sample in ST

Since the table, from which the records are to be read, has the structure below, a PLC structure with a similar structure must be created.

Table:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Bigint</td>
</tr>
<tr>
<td>Timestamp</td>
<td>datetime</td>
</tr>
<tr>
<td>Name</td>
<td>nvarchar(80)</td>
</tr>
<tr>
<td>Value</td>
<td>float</td>
</tr>
</tbody>
</table>
Structure:

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

The library TcUtilities.lib must be integrated in order to be able to use the data type T_ULARGE_INTEGER.

For ARM processors the data types have to be arranged differently due to the byte alignment, and a "dummy byte" has to be added.

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

**Program MAIN**

```plaintext
VAR
  FB_DBRecordArraySelect1 : FB_DBRecordArraySelect;
  cmd := '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**

```plaintext
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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
6.2.1.18  **FB_DBStoredProcedures**

<table>
<thead>
<tr>
<th>FB_DBSTOREDPROCEDURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetID: T_AmsNetID</td>
</tr>
<tr>
<td>hDBID: UDINT</td>
</tr>
<tr>
<td>sProcedureName: T_MaxString</td>
</tr>
<tr>
<td>cbParameterList: UDINT</td>
</tr>
<tr>
<td>pParameterList: POINTER TO ARRAY[0..MAX_STORED_PROCEDURES_PARAMETERS] OF ST_DBParameter</td>
</tr>
<tr>
<td>bExecute: BOOL</td>
</tr>
<tr>
<td>tTimeout: TIME</td>
</tr>
<tr>
<td>bBusy: BOOL</td>
</tr>
<tr>
<td>bError: BOOL</td>
</tr>
<tr>
<td>nErrID: UDINT</td>
</tr>
<tr>
<td>sSQLState: ST_DBSQLError</td>
</tr>
</tbody>
</table>

The function block FB_DBStoredProcedures can be used to call up stored procedures. They can include parameters in the process, which are used in the stored procedures.

**VAR_INPUT**

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

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
- **hDBID**: Indicates the ID of the database to be used.
- **sProcedureName**: Indicates the name of the procedure to be executed.
- **cbParameterList**: Indicates the length of the parameter list.
- **pParameterList**: Contains the address of the parameter list.
- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

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

- **bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
- **bError**: Becomes TRUE, as soon as an error occurs.
- **nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.
- **sSQLState**: Returns the SQL error code [320] of the corresponding database type.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
6.2.1.19  FB_DBStoredProceduresRecordArray

The function block FB_DBStoredProceduresRecordArray can be used to call stored procedures that return records. In contrast to the FB_DBStoredProceduresRecordReturn function block, this function block can be used to return several records with a single call. They can include parameters in the process, which are used in the stored procedures.

VAR_INPUT

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

sProcedureName: Indicates the name of the procedure to be executed.

cbParameterList: Indicates the length of the parameter list.

pParameterList: Contains the address of the parameter list

nStartIndex: Indicates the index of the first record to be read.

nRecordCount: Indicates the number of records to be read.

cbRecordArraySize: Indicates the size of the structure array in bytes.

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

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the time before the function is cancelled.

VAR_OUTPUT

VAR_OUTPUT
bBusy : BOOL;
setError : BOOL;
nErrID : UDINT;
sSQLState : ST_DBSQLError;
nRecords : UDINT;
END_VAR

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as "bBusy" remains TRUE.
bError: Becomes TRUE, as soon as an error occurs.

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [p. 399] if the bError output is set.

sSQLState: Returns the SQL error code [p. 320] of the corresponding database type

nRecords: Returns the number of data records.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1.20 Obsolete

6.2.1.20.1 FB_DBAuthenticationAdd

The function block FB_DBAuthenticationAdd permits authentication information of declared database connection to be added to the XML configuration file or to be changed.

**VAR_INPUT**

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

sNetID: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

sDBSystemDB: Only for Access databases. Indicates the path to the MDW file.

sDBUserId: Indicates the login user name.

sDBPassword: Indicates the password.

bExecute: The command is executed with a rising edge.

tTimeout: Indicates the duration of the timeout.

**VAR_OUTPUT**

```plaintext
VAR_OUTPUT
  bBusy     : BOOL;
  bError    : BOOL;
  nErrID    : UDINT;
END_VAR
```
**bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as “bBusy” remains TRUE.

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

**nErrID**: Returns the ADS error code if the bError output is set.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

### 6.2.1.20.2 FB_DBRecordInsert

The function block FB_DBRecordInsert can be used to write individual records with any structure into a database. To use the function block it is necessary to declare the database to which the records are to be written in the XML configuration file.

**VAR_INPUT**

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

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
- **hDBID**: Indicates the ID of the database to be used.
- **sInsertCmd**: Indicates which INSERT command is to be executed.
- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

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

- **bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
- **bError**: Becomes TRUE, as soon as an error occurs.
- **nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.
- **sSQLState**: Returns the SQL error code [320] of the corresponding database type.
Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.1.20.3 **FB_DBRecordSelect**

The **FB_DBRecordSelect** allows individual data records to be read from a database.

This function block is not compatible with ASCII files.

**VAR_INPUT**

```plaintext
VAR_INPUT
  sNetID   : T_AmsNetID;
  hDBID    : UDINT;
  sSelectCmd : T_MaxString;
  nRecordIndex : UDINT;
  cbRecordSize : UDINT;
  pDestAddr : DWORD;
  bExecute : BOOL;
  tTimeout : TIME;
END_VAR
```

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
- **hDBID**: Indicates the ID of the database to be used.
- **sSelectCmd**: Indicates which SELECT command is to be executed.
- **nRecordIndex**: Gives the index of the data record that is to be read.
- **cbRecordSize**: Provides the size of a data record in bytes.
- **pDestAddr**: Indicates the address of the structure to which the record is to be written.
- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

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

- **bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
- **bError**: Becomes TRUE, as soon as an error occurs.
- **nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.
**sSQLState**: Returns the SQL error code [320] of the corresponding database type.

**nRecords**: Returns the number of data records.

**Example in ST**:

Since the table, from which the records are to be read, has the structure below, a PLC structure with a similar structure must be created.

**Table**:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Bigint</td>
</tr>
<tr>
<td>Timestamp</td>
<td>datetime</td>
</tr>
<tr>
<td>Name</td>
<td>Ntext</td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
</tr>
</tbody>
</table>

**Structure**:

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

The library TcUtilities.lib must be integrated in order to be able to use the data type T_ULARGE_INTEGER.

For ARM processors the data types have to be arranged differently due to the byte alignment, and a "dummy BYTE" has to be added.

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

**PLC program**

```plaintext
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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

#### 6.2.1.20.4 FB_DBRecordSelect_EX

The function block FB_DBRecordSelect_EX can be used to read individual records with any structure from the database. This function block can be used to execute an SQL SELECT command with up to 10,000 characters.

This function block is not compatible with ASCII files.

**VAR_INPUT**

```plaintext
VAR_INPUT
    sNetID : T_AmsNetID;
    Hdbid : UDINT;
    cbCmdSize : UDINT;
    pCmdAddr : UDINT;
    nRecordIndex : UDINT;
    cbRecordSize : UDINT;
    pDestAddr : POINTER TO BYTE;
    bExecute : BOOL;
    tTimeout : TIME;
END_VAR
```

- **sNetID**: String containing the AMS network ID of the target device, at which the ADS command is directed.
- **hDBID**: Indicates the ID of the database to be used.
- **cbCmdSize**: Indicates the length of a SELECT command to be executed.
- **pCmdSize**: Indicates the pointer address of a string variable with the SQL command to be executed.
- **nRecordIndex**: Gives the index of the data record that is to be read.
- **cbRecordSize**: Provides the size of a data record in bytes.
- **pDestAddr**: Indicates the address of the structure to which the record is to be written.
- **bExecute**: The command is executed with a rising edge.
- **tTimeout**: Indicates the time before the function is cancelled.

**VAR_OUTPUT**

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

- **bBusy**: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.
**bError**: Becomes TRUE, as soon as an error occurs.

**nErrID**: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.

**sSQLState**: Returns the SQL error code [320] of the corresponding database type

**nRecords**: Returns the number of data records.

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

### 6.2.1.20.5 FB_DBStoredProceduresRecordReturn

The function block FB_DBStoredProceduresRecordReturn can be used to call up stored procedures that return a record. They can include parameters in the process, which are used in the stored procedures.

**VAR_INPUT**

```plaintext
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**: String containing the AMS network ID of the target device, at which the ADS command is directed.

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

**sProcedureName**: Indicates the name of the procedure to be executed.

**cbParameterList**: Indicates the length of the parameter list.

**pParameterList**: Contains the address of the parameter list

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

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

**pRecordAddr**: Indicates the address of the structure to which the record is to be written.

**bExecute**: The command is executed with a rising edge.

**tTimeout**: Indicates the time before the function is cancelled.
VAR_OUTPUT

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

bBusy: The command is in the process of being transmitted by ADS. No new command will be accepted as long as bBusy remains TRUE.

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

nErrID: Returns the ADS error code or TcDatabaseSrv_Error_Codes [399] if the bError output is set.

sSQLState: Returns the SQL error code [320] of the corresponding database type

nRecords: Returns the number of data records.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target system type</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.2 Data types

6.2.2.1 ST_DBColumnCfg

VAR_INPUT

TYPE ST_DBColumnCfg:
  STRUCT
    sColumnName: STRING(59);
    sColumnProperty: STRING(59);
    eColumnType: E_DbColumnTypes;
  END_STRUCT
END_TYPE

sColumnName: Contains the name of the column to be created.

sColumnProperty: Contains certain column properties.

eColumnType: Gives the type of column.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.2.2 ST_DBXMLCfg

VAR_INPUT

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

sDBName: Contains the name of the database.
PLC API

**sDBTable:** Contains the name of the table.

**nDBID:** Returns the ID of the database.

**eDBType:** Gives the type of database.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

#### 6.2.2.3 ST_ADSDevXMLCfg

**VAR_INPUT**

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

**sAdsDevNetID:** String containing the AMS network ID of the ADS device.

**tAdsDevTimeout:** Indicates the timeout time of the ADS device.

**nAdsDevID:** Returns the ID of the ADS device.

**nAdsDevPort:** Indicates the port of the ADS device.

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

#### 6.2.2.4 ST_DBSQLError

**VAR_INPUT**

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

**sSQLState:** Contains the 5-character error code, which is based on the SQL ANSI standard.

**nSQLErrorCode:** Returns a database-specific error code.

If no error has occurred, the structure contains the following values:

- `sSQLState := '00000';`
- `nSQLErrorCode := 0;`

### Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
### 6.2.2.5 ST_DBParameter

**VAR_INPUT**

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

- **sParameterName**: Indicates the name of the parameter.
- **cbParameterValue**: Contains the size of the variable to be used in bytes.
- **pParameterValue**: Contains the address of the variable to be used.
- **eParameterDataType**: Indicates the data type of the parameter (E_DbColumnTypes [321]).
- **eParameterType**: Indicates the parameter type (E_DBParameterTypes [323]).

**Declaration sample**

**Variable Declaration**

```plaintext
PROGRAM MAIN
VAR
  paraList: ARRAY [0..2] OF ST_DBParameter;
p1: DINT := 3;
p2: LREAL;
p3: STRING;
END_VAR

PLC program
```

```plaintext
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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

### 6.2.2.6 E_DbColumnTypes

```plaintext
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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.2.7 E_DBTypes

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

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.2.8 E_DBValueType

TYPE E_DBValueType :
{
    eDBValue_Double := 0,
eDBValue_Bytes := 1
};
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

6.2.2.9 E_DBWriteModes

TYPE E_DBWriteModes :
{
    eDBWriteMode_Update := 0,
eDBWriteMode_Append := 1,
eDBWriteMode_RingBuffer_Time := 2,
eDBWriteMode_RingBuffer_Count := 3
};
END_TYPE

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
Requirements

Development environment | Target platform | PLC libraries to be linked
--- | --- | ---
TwinCAT v3.0.0 | PC or CX (x86) | Tc2_Database

6.2.2.10 E_DBParameterTypes

```haskell
TYPE E_DBParameterTypes :
{
  eDBParameter_Input := 0,
  eDBParameter_Output := 1,
  eDBParameter_InputOutput := 2,
  eDBParameter_ReturnValue := 3,
  eDBParameter_OracleCursor := 4
};
END_TYPE
```

Requirements

Development environment | Target platform | PLC libraries to be linked
--- | --- | ---
TwinCAT v3.0.0 | PC or CX (x86) | Tc2_Database

6.2.3 Global Constants

6.2.3.1 Constants

```haskell
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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

### 6.2.3.2 Library version

All libraries have a certain version. The version is indicated in the PLC library repository, for example. A global constant contains the information about the library version:

**Global_Version**

```plaintext
VAR_GLOBAL CONSTANT
  stLibVersion_TC3_Database_Server : ST_LibVersion;
END_VAR
```

Use the function `F_CmpLibVersion` (defined in the Tc2_System library) to check whether you are using the correct version.

- All other options for comparing library versions, which you may know from TwinCAT 2, are outdated!
7 Examples

7.1 Tc3_Database

The following pages introduce samples and tips for using the TwinCAT Database Server with the library for TwinCAT 3.

7.1.1 Scenario examples

Each sample is assigned to a scenario, which can be adapted to specific use cases. Information on which databases the samples are compatible with is also provided.

7.1.1.1 Home automation

This scenario example illustrates the configuration mode for building automation. Symbols are written to an MySQL database in three different AutoLog groups without additional programming, i.e. purely based on the configuration. Room temperatures are logged in the database at 5-minute intervals. Energy data are saved at 1-minute intervals. Events such as "lamp on", "lamp off" are stored.

<table>
<thead>
<tr>
<th>Category</th>
<th>Configure Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database used</td>
<td>MySQL</td>
</tr>
<tr>
<td>Compatible databases</td>
<td>Can be used for all supported database types</td>
</tr>
<tr>
<td>PLC function blocks used</td>
<td>None</td>
</tr>
<tr>
<td>PLC libraries used</td>
<td>Tc3_Database, Tc2_BABasic</td>
</tr>
<tr>
<td>Download</td>
<td>[Download Link]</td>
</tr>
</tbody>
</table>

Step 1: Once a new TwinCAT Database Server project was created, the database connection is added and configured.

[Download Link]
Step 2: Since logging is to take place in three different tables, three AutoLog groups are appended to the database connection. First the temperature AutoLog group is configured. The CycleTime is set to 300000 ms to match the 5-minute database logging interval.

Step 3: Then the ADS device from which the ADS symbols are to be read is set up.
Step 4: The symbols are created with the target browser.

Step 5: Now you can select the table into which the temperature values are to be written. The AutoLog groups support two TableModes [39]. The standard table structure is used for the temperature values. A tick indicates that the selected table has the correct structure.
Step 6: Then the AutoLog group for the events is configured. The LogMode for the group is set to "onChange", the cycle time to 1000 ms. This means that the symbols are checked at 1-second intervals but a table entry is only created if a value changes.

Step 7: Then the ADS device and the symbols are selected, as described in steps 3 and 4.
Step 8: The events are also stored in a standard table structure.

Step 9: Finally, the energy data are saved at 1-minute intervals.
Step 10: For the energy data the ADS device and the symbols are also selected.

Step 11: The energy data should be stored in a custom table structure, with the various symbols assigned to the columns.
Step 12: Logging can now be started and monitored with the AutoLog Viewer [50].

Documents about this
TF6420_Sample1_Homeautomation.zip (Resources/zip/9007202749926411.zip)

7.1.1.2 Message logger

This scenario example illustrates the PLC Expert Mode for a Message Logger in the PLC. In the sample program the function blocks of the TwinCAT Database Server are used to create a function block, which provides various methods for generating and reading messages. The database in which the messages are stored is created from the PLC. A sample application of the created function block is implemented in the MAIN program. A new database file is created every 7 days. Three different messages can be sent. In addition it is possible to call up the last message or all messages from a particular interval.
**Category** | PLC Expert Mode  
---|---  
**Database used** | MS Compact  
**Compatible databases** | Can be used with minor amendments for all supported database types  
**PLC function blocks used** | FB_PLCDBCreateEvt [168], FB_PLCDBCmdEvt [180], FB_PLCDBWriteEvt [175], FB_PLCDBReadEvt [171]  
**PLC libraries used** | Tc3_Database, Tc3_Eventlogger  
**Download** | [Download](https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/9007202749928971.zip)  

### FB_ErrorLogger

**CreateErrorLogDB (method)**

The CreateErrorLogDB method creates an MS Compact database file and the table in which the messages are stored.

```plaintext
METHOD CreateErrorLogDB : BOOL  
VAR_INPUT  
  sDBName : T_MaxString;  
END_VAR  
CASE nState_CreateDB OF  
  0:  
    stDBConfig.sServer := CONCAT(CONCAT(sDBPath, sDBName), '.sdf');  
    stDBConfig.bAuthentication := FALSE;  
    nState_CreateDB := 1;  
  1:  
    IF fbPLCDBCreate.Database(pDatabaseConfig:= ADR(stDBConfig), cbDatabaseConfig:= SIZEOF(stDBConfig), bCreateXMLConfig:= FALSE, pDBID:= 0) THEN  
      ipResultEvt := fbPLCDBCreate.ipTcResult;  
      IF fbPLCDBCreate.bError THEN  
        nState_CreateDB := 100;  
      ELSE
```
nState_CreateDB := 2;
END_IF

2:
IF fbConfigTcDBSrv.Create(pTcDBSrvConfig:= ADR(stDBConfig),
cbTcDBSrvConfig:= SIZEOF(stDBConfig), bTemporary:= TRUE,
pConfigID:= ADR(nDBID) ) THEN
  ipResultEvt := fbConfigTcDBSrv.ipTcResult;
  IF fbConfigTcDBSrv.bError THEN
    nState_CreateDB := 100;
  ELSE
    nState_CreateDB := 3;
  END_IF
END_IF

3:
arrTableColumns[0].sName := 'ID';
arrTableColumns[0].eType := E_ColumnType.BigInt;
arrTableColumns[0].nLength := 8;
arrTableColumns[0].sProperty := 'IDENTITY(1,1)';
arrTableColumns[1].sName := 'Timestamp';
arrTableColumns[1].eType := E_ColumnType.DateTime;
arrTableColumns[1].nLength := 4;
arrTableColumns[2].sName := 'Severity';
arrTableColumns[2].eType := E_ColumnType.NVarChar;
arrTableColumns[2].nLength := 10;
arrTableColumns[3].sName := 'ErrorCode';
arrTableColumns[3].eType := E_ColumnType.Integer;
arrTableColumns[3].nLength := 4;
arrTableColumns[4].sName := 'Message';
arrTableColumns[4].eType := E_ColumnType.NVarChar;
arrTableColumns[4].nLength := 255;
IF fbPLCDBCreate.Table(hDBID:= nDBID, sTableName:= sTableName,
pTableCfg:= ADR(arrTableColumns),
cbTableCfg:= SIZEOF(arrTableColumns)) THEN
  ipResultEvt := fbPLCDBCreate.ipTcResultEvent;
  nState_CreateDB := 100;
END_IF

100:
IF _SetResultInfo(1033) THEN
  IF NOT bError THEN
    _bHasCreated := TRUE;
  END_IF
  nState_CreateDB := 0;
END_IF
END_CASE

CreateErrorLogDB := nState_CreateDB = 0;

AddErrorEntry (method)
The AddErrorEntry method can be used to write different messages into the database.
METHOD AddErrorEntry : BOOL
VAR_INPUT
  tTimestamp : DT;
  eSeverity : E_Severity;
  nErrCode : UDINT;
  sMessage : T_MaxString;
END_VAR
CASE nState_AddEntry OF
  0:
    ipResultEvt := fbPLCDBWrite.ipTcResult;
    stError.tTimestamp := tTimestamp;
    CASE eSeverity OF
      TcEventSeverity.Info:
        stError.sSeverity := 'Info';
      TcEventSeverity.Warning:
        stError.sSeverity := 'Warning';
      TcEventSeverity.Verbose:
        stError.sSeverity := 'Verbose';
      TcEventSeverity.Critical:
        stError.sSeverity := 'Critical';
TcEventSeverity.Error:
  stError.sSeverity := 'Error';
END_CASE
stError.nErrCode := nErrCode;
stError.sMsg := sMessage;
arrColumns[0] := 'Timestamp';
nState_AddEntry := 1;
1:
  IF fbPLCDBWrite.WriteStruct(  
hDBID:= nDBID,
sTableName:= sTableName,
pRecord:= ADR(stError),
cbRecord:= SIZEOF(stError),
pColumnNames:= ADR(arrColumns),
cbColumnNames:= SIZEOF(arrColumns)) THEN
    nState_AddEntry := 100;
  END_IF
100:
  IF _SetResultInfo(1033) THEN
    nState_AddEntry := 0;
  END_IF
END_CASE
AddErrorEntry := nState_AddEntry = 0;

ReadLastError (method)
The method ReadLastError can be used to read the latest (last) entry from the database.

METHOD ReadLastError : BOOL
VAR_OUTPUT
  tTimestamp: DT;
sSeverity: STRING(10);
nErrCode: UDINT;
sMessage: T_MaxString;
END_VAR
CASE nState_ReadLastEntry OF
  0:
    ipResultEvt := fbPLCDBRead.ipTcResult;
arrColumns[0] := 'Timestamp';

  nState_ReadLastEntry := 1;
1:
  IF fbPLCDBRead.ReadStruct(  
hDBID:= nDBID,
sTableName:= sTableName,
pColumnNames:= ADR(arrColumns),
cbColumnNames:= SIZEOF(arrColumns),
sOrderByColumn:= 'ID',
eOrderType:= E_OrderType.DESC,
nStartIndex:= 0,
nRecordCount:= 1,
pData:= ADR(stReadData),
cbData:= SIZEOF(stReadData)) THEN
    nState_ReadLastEntry := 100;
  END_IF
100:
  IF _SetResultInfo(1033) THEN
    IF NOT fbPLCDBRead.bError THEN
      tTimestamp := stReadData.tTimestamp;
      sSeverity := stReadData.sSeverity;
      nErrCode := stReadData.nErrCode;
      sMessage := stReadData.sMsg;
    END_IF
    nState_ReadLastEntry := 0;
  END_IF
Examples

ReadLastError := nState_ReadLastEntry = 0;

**GetErrorTimerange (method)**

The method GetErrorTimerange can be used to read all messages from a particular interval.

```
METHOD GetErrorTimerange : BOOL
VAR_INPUT
  tStartTimestamp : DT;
  tEndTimestamp : DT;
  nStartIndex : UDINT;
END_VAR
VAR_OUTPUT
  nErrorCount: UDINT;
  arrErrors : ARRAY [0..10] OF ST_ErrorEntry;
END_VAR
CASE nState_ErrorTimerange OF
  0:
    ipResultEvt := fbPLCDBRead.ipTcResult;
    stSearchData.dtStartTimestamp := tStartTimestamp;
    stSearchData.dtEndTimestamp := tEndTimestamp;
    sCmd := 'SELECT Timestamp, ErrorCode, Severity, Message FROM tbl_Errors WHERE Timestamp >= {start} AND Timestamp <= {end}';
    arrParameter[0].sParaName := 'start';
    arrParameter[0].eParaType := E_ExpParameterType.DateTime;
    arrParameter[0].nParaSize := 4;
    arrParameter[1].sParaName := 'end';
    arrParameter[1].eParaType := E_ExpParameterType.DateTime;
    arrParameter[1].nParaSize := 4;
    nState_ErrorTimerange := 1;
  1:
    IF fbPLCDBCmd.ExecuteDataReturn(
      hDBID:= nDBID,
      pExpression:= ADR(sCmd),
      cbExpression:= SIZEOF(sCmd),
      pData:= ADR(stSearchData),
      cbData:= SIZEOF(stSearchData),
      pParameter:= ADR(arrParameter),
      cbParameter:= SIZEOF(arrParameter),
      nStartIndex:= nStartIndex,
      nRecordCount:= 10,
      pReturnData:= ADR(arrErrs),
      cbReturnData:= SIZEOF(arrErrs),
      pRecords:= ADR(nErrCount)) THEN
      nState_ErrorTimerange := 100;
      END_IF
  100:
    IF _SetResultInfo(1033) THEN
      nErrorCount := nErrCount;
      arrErrors := arrErrs;
      nState_ErrorTimerange := 0;
      END_IF
END_CASE
GetErrorTimerange := nState_ErrorTimerange = 0;

_setResultInfo (private method)

The I_Message message interface is evaluated by the TwinCAT EventLogger in the private _SetResultInfo method.

```
METHOD _SetResultInfo : BOOL
VAR_INPUT
  nLangId : INT := 1033;
END_VAR
_setResultInfo := FALSE;
CASE nState_SetResInfo OF
```
```
0: IF ipResultEvt.RequestEventText(nLangId, EventText, SIZEOF(EventText)) THEN
    nState_SetResInfo := 1;
END_IF
1: IF ipResultEvt.RequestEventClassName(nLangId, EventClassName, SIZEOF(EventClassName)) THEN
    EventSourcePath := ipResultEvt.ipSourceInfo.sName;
    EventId := ipResultEvt.nEventId;
    bError := (ipResultEvt.eSeverity = TcEventSeverity.Error) OR
       (ipResultEvt.eSeverity = TcEventSeverity.Critical);
    nState_SetResInfo:=0;
    _SetResultInfo := TRUE;
END_IF
END_CASE
```

Documents about this

TF6420_Sample2_ErrorLogger.zip (Resources/zip/9007202749928971.zip)

### 7.1.1.3 Production register

This scenario example illustrates the use of the SQL Expert Mode for handling stored procedures. A connection to the database established from the PLC. A stored procedure is used to read product positions from several tables. A visualization is used for the operation.

<table>
<thead>
<tr>
<th>Category</th>
<th>SQL Expert mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database used</td>
<td>MS SQL [124]</td>
</tr>
<tr>
<td>Compatible databases</td>
<td>MS SQL, MySQL, Oracle</td>
</tr>
<tr>
<td>PLC function blocks used</td>
<td>FB_SQLDatabaseEvt [188], FB_SQLStoredProcedureEvt [198], FB_SQLResultEvt [195]</td>
</tr>
<tr>
<td>PLC libraries used</td>
<td>Tc3_Database, Tc3_Eventlogger</td>
</tr>
<tr>
<td>Download</td>
<td><a href="https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/18014402004672523.zip">https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/18014402004672523.zip</a></td>
</tr>
</tbody>
</table>
In the MAIN program a so-called state machine is implemented for processing, through which the different SQL function blocks are controlled. Since the methods of the function blocks no longer have an Execute flag, the user must ensure that the method is not called again in the next cycle, in order to avoid repetition of the procedure. This can easily be ensured through the state machine.

PROGRAM MAIN
VAR
  bCONNECT: BOOL;
  bEXECUTE: BOOL;
  bREAD : BOOL;
  bDISCONNECT: BOOL;

  R_TRIG1: R_TRIG;
  R_TRIG2: R_TRIG;
  R_TRIG3: R_TRIG;
  R_TRIG4: R_TRIG;

  nState: INT;
  nState_Connect: INT;
  nState_Disconnect: INT;

  bConn: BOOL;
  bSP: BOOL;
  bResult: BOOL;
  bData: BOOL;

  nDBID: UDINT := 1;

  fbSQLDatabase: FB_SQLDatabaseEvt(sNetID='', tTimeout:=T#10S);
  fbSQLStoredProcedure: FB_SQLStoredProcedureEvt(
    sNetID:='', tTimeout:=T#10S);
  fbSQLResult: FB_SQLResultEvt(sNetID:='', tTimeout:=T#10S);

  arrParameter: ARRAY [0..0] OF ST_SQLSPParameter;

  nCustomerID: DINT := 12345;
  nRecordStartIndex: UDINT;
  stRecordArr: ARRAY [1..20] OF ST_Record;
  nRecs: UDINT;
  ipResultEvt : Tc3_Eventlogger.I_TcMessage;
  bError : BOOL;
  nEventID: UDINT;
  sEventClass : STRING(255);
  sEventMsg : STRING(255);
END_VAR

R_TRIG1(CLK:=bCONNECT);
IF R_TRIG1.Q AND nState = 0 THEN
  nState := 1;
END_IF

R_TRIG2(CLK:=bEXECUTE);
IF R_TRIG2.Q AND nState = 0 THEN
  nState := 2;
END_IF

R_TRIG3(CLK:=bREAD);
IF R_TRIG3.Q AND nState = 0 THEN
  nState := 3;
END_IF

R_TRIG4(CLK:=bDISCONNECT);
IF R_TRIG4.Q THEN
  nState := 4;
END_IF

CASE nState OF
  0: (*Idle*)
  1: // Connect to database and create stored procedure instance
      CASE nState_Connect OF
        0: IF fbSQLDatabase.Connect(hDBID:= nDBID) THEN
            ipResultEvt := fbSQLDatabase.ipTcResult;
           bConn := NOT fbSQLDatabase.bError;
           IF bConn THEN
             nState_Connect := 1;
         END_IF
       END_CASE
END_CASE

Examples

nState:=200;
END_IF
END_IF
1:
arrParameter[0].sParameterName := '@Customer_ID';
arrParameter[0].eParameterDataType := Tc3_Database.E_ColumnType.Integer;
arrParameter[0].eParameterType := E_SPParameterType.Input;
arrParameter[0].nParameterSize := SIZEOF(nCustomerID);

IF fbSQLDatabase.CreateSP('SP_GetAddressByCustomerID', ADR(arrParameter), SIZEOF(arrParameter), ADR(fbSQLStoredProcedure)) THEN
ipResultEvt:= fbSQLDatabase.ipTcResult;
bSP := NOT fbSQLDatabase.bError;
nState_Connect:=0;
nState := 200;
END_IF
END_CASE
2: // Execute stored procedure
IF fbSQLStoredProcedure.ExecuteDataReturn( pParameterStrc:= ADR(nCustomerID), cbParameterStrc:= SIZEOF(nCustomerID), pSQLDBResult:= ADR(fbSQLResult)) THEN
ipResultEvt:= fbSQLStoredProcedure.ipTcResult;
MEMSET(ADR(stRecordArr),0,SIZEOF(stRecordArr));
bResult := NOT fbSQLStoredProcedure.bError;
nState := 200;
END_IF
END_IF
3: // Read customer positions
IF fbSQLResult.Read(nRecordStartIndex, 20, ADR(stRecordArr), SIZEOF(stRecordArr), TRUE, FALSE) THEN
ipResultEvt:= fbSQLResult.ipTcResult;
bData := NOT fbSQLStoredProcedure.bError;
nRecs := fbSQLResult.nDataCount;
nState := 200;
END_IF
END_IF
4:// Disconnect all
CASE nState_Disconnect OF
0:
IF bData THEN
  IF fbSQLResult.Release() THEN
    nState_Disconnect := 1;
  END_IF
ELSE
  nState_Disconnect := 1;
END_IF
1:
IF bSP THEN
  IF fbSQLStoredProcedure.Release() THEN
    nState_Disconnect := 2;
  END_IF
ELSE
  nState_Disconnect := 2;
END_IF
2:
IF bConn THEN
  IF fbSQLDatabase.Disconnect() THEN
    nState_Disconnect := 3;
  END_IF
ELSE
  nState_Disconnect := 3;
END_IF
3:
bData := FALSE;
bSP := FALSE;
bConn := FALSE;
bResult := FALSE;
sEventClass := "";
sEventMsg := "";
nEventID := 0;
bError := FALSE;
nState_Disconnect := 0;
nState := 0;
END_CASE
200:
IF ipResultEvt.RequestEventText(1033, sEventMsg, SIZEOF(sEventMsg)) THEN
  nState := 201;
END_IF
The individual process steps can be reproduced in the individual PLC states. Boolean flags are available to facilitate handling.

1. bConnect: Connection with the database is established
2. bExecute: The stored procedure is executed, and results are loaded into the cache
3. bRead: The results are transferred to the PLC
4. bDisconnect: The connection is closed

If these steps are executed consecutively, the array stRecordArr is filled with values from the database:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
<th>Prepared value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bConnect</td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>bExecute</td>
<td>BOOL</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>bRead</td>
<td>BOOL</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>bDisconnect</td>
<td>BOOL</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>stRecordArr</td>
<td>ARRAY</td>
<td>ARRAY [1..20] OF S...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stRecord</td>
<td>ST_Record</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nID</td>
<td>ULINT 12345</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sCustomer</td>
<td>STRING(50) Johann'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sName</td>
<td>STRING(50) Gm, '</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nProductNum</td>
<td>DINT 50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sProductName</td>
<td>STRING(50) CX100D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sProductInfo</td>
<td>T_MixedString E'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CTimestamp</td>
<td>DATE_AND_TIME DT2010-10-10-...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stRecordArr[2]</td>
<td>ST_Record</td>
<td></td>
</tr>
<tr>
<td></td>
<td>stRecordArr[3]</td>
<td>ST_Record</td>
<td></td>
</tr>
</tbody>
</table>

Documents about this
TF6420_Sample3_CustomerProducts.zip (Resources/zip/18014402004672532.zip)

7.1.1.4 Production recipe

This scenario example illustrates how the TwinCAT Database Server handles XML files with any structure. The production recipe for building the product is read from an XML file. The corresponding test parameters are read from a different file. In addition, the test results are written into an existing XML file.
### Examples

**Category** | SQL Expert mode  
**Database used** | XML (as free XML documents)  
**Compatible databases** | XML  
**PLC function blocks used** | FB_PLCDBCmdEvt  
**PLC libraries used** | Tc3_Database, Tc3_Eventlogger  
**Download** | [Download Link](https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/9007202749934091.zip)  

Recipe XML:
Examples

Test XML:

```xml
<ProductionConfig>
  <Config TypeNum="12345" Test="985-524">
    <rLength>65.85</rLength>
    <rWidth>30</rWidth>
    <rHeight>2.5</rHeight>
    <iQuantity>500</iQuantity>
    <iCounter>0</iCounter>
  </Config>
  <Config TypeNum="23456" Test="985-524">
    <rLength>15.85</rLength>
    <rWidth>300</rWidth>
    <rHeight>12.5</rHeight>
    <iQuantity>1200</iQuantity>
    <iCounter>0</iCounter>
  </Config>
  <Config TypeNum="34567" Test="125-594">
    <rLength>195.85</rLength>
    <rWidth>378</rWidth>
    <rHeight>145.5</rHeight>
    <iQuantity>10</iQuantity>
    <iCounter>0</iCounter>
  </Config>
</ProductionConfig>
```

FB_ProductionConfigData

GetConfig (method)

This method reads the production recipe for a product from an XML file. XPath queries can be used to find the required recipe.

```delphi
METHOD GetConfig : BOOL
VAR_INPUT
    nTypeNum : DINT;
END_VAR
VAR_OUTPUT
    stConfig : ST_Config;
END_VAR
GetConfig := FALSE;
arrPara[0].sParaName := 'rLength';
arrPara[0].eParaType := Tc3_Database.E_ExpParameterType.Float32;
arrPara[0].nParaSize := 4;
arrPara[1].sParaName := 'rWidth';
arrPara[1].eParaType := Tc3_Database.E_ExpParameterType.Float32;
```
GetTestParameter (method)

This method reads the product-specific test parameters.

METHOD GetTestParameter : BOOL
VAR_INPUT
  nTypeNum : DINT;
END_VAR
VAR_OUTPUT
  stTestPara: ST_TestParameter;
END_VAR
GetTestParameter := FALSE;
CASE nState_GetTestPara OF
  0:
    IF fbPLCDCmd.ExecuteDataReturn(
      hDBID:= 1,
      pExpression:= ADR(sCmd),
      cbExpression:= SIZEOF(sCmd),
      pData:= 0,
      cbData:= 0,
      pParameter:= ADR(arrPara),
      cbParameter:= SIZEOF(arrPara[0]),
      nStartIndex:= 0,
      nRecordCount:= 1,
      pReturnData:= ADR(_stConfig),
      cbReturnData:= SIZEOF(_stConfig),
      pRecords:= 0) THEN
      sTestNum := CharsOf(arrPara[0]).To_CString;
    END_IF
  100:
    IF _SetResultInfo(1033) THEN
      GetConfig := TRUE;
      nState_GetTestPara := 0;
    END_IF
END_CASE

arrPara[1].nParaSize := 4;
arrPara[2].sParaName := 'rHeight';
arrPara[2].eParaType := Tc3_Database.E_ExpParameterType.Float32;
arrPara[2].nParaSize := 4;
arrPara[3].sParaName := 'iQuantity';
arrPara[3].eParaType := Tc3_Database.E_ExpParameterType.Int32;
arrPara[3].nParaSize := 4;
arrPara[4].sParaName := 'iCounter';
arrPara[4].eParaType := Tc3_Database.E_ExpParameterType.Int32;
arrPara[4].nParaSize := 4;
sCmd := CONCAT(CONCAT('XPATH_SEL<ATTR>#ProductionConfig/Config[@TypeNum
  = ', DINT_TO_STRING(nTypeNum), '), ''']);

CASE nState_GetConfig OF
  0:
    IF fbPLCDCmd.ExecuteDataReturn(
      hDBID:= 1,
      pExpression:= ADR(sCmd),
      cbExpression:= SIZEOF(sCmd),
      pData:= 0,
      cbData:= 0,
      pParameter:= ADR(arrPara),
      cbParameter:= SIZEOF(arrPara[0]),
      nStartIndex:= 0,
      nRecordCount:= 1,
      pReturnData:= ADR(_stConfig),
      cbReturnData:= SIZEOF(_stConfig),
      pRecords:= 0) THEN
      ipResultEvt := fbPLCDCmd.ipTcResult;
      nState_GetConfig := 100;
    END_IF
  100:
    IF _SetResultInfo(1033) THEN
      GetConfig := TRUE;
      stConfig := _stConfig;
      nState_GetConfig := 0;
    END_IF
END_CASE
```
1:  
   arrPara[0].sParaName := 'MaxTemp';
   arrPara[0].eParaType := Tc3_Database.E_ExpParameterType.Float32;
   arrPara[0].nParaSize := 4;

   arrPara[1].sParaName := 'MinTemp';
   arrPara[1].eParaType := Tc3_Database.E_ExpParameterType.Float32;
   arrPara[1].nParaSize := 4;

   arrPara[2].sParaName := 'MaxPSI';
   arrPara[2].eParaType := Tc3_Database.E_ExpParameterType.Int32;
   arrPara[2].nParaSize := 4;

   sCmd := CONCAT(CONCAT('XPATH_SEL<SUBTAG>#ProductionConfig/
     TestParameter/Test[@Num = "$''', _sTestNum), '$']));

   IF fbPLCDBCmd.ExecuteDataReturn(  
      hDBID:= 2,
      pExpression:= ADR(sCmd),
      cbExpression:= SIZEOF(sCmd),
      pData:= 0,
      cbData:= 0,
      pParameter:= ADR(arrPara),
      cbParameter:= SIZEOF(arrPara[0])*3,
      nStartIndex:= 0,
      nRecordCount:= 1,
      pReturnData:= ADR(_stTest),
      cbReturnData:= SIZEOF(_stTest),
      pRecords:= 0) THEN
      ipResultEvt := fbPLCDBCmd.ipTcResult;
      nState_GetTestPara := 100;
   ELSE
      nState_GetTestPara := 1;
   END_IF
END_IF
```

AddTestEntry (method)

This method adds the test result to the test XML file.

```
METHOD AddTestEntry : BOOL
VAR_INPUT  
   sTestNum : STRING(8);
   nTypeNum : DINT;
   sTimestamp : STRING;
   sTester : STRING;
   sResult : STRING;
END_VAR
AddTestEntry := FALSE;

   arrPara[0].sParaName := 'TestNum';
   arrPara[0].eParaType := Tc3_Database.E_ExpParameterType.STRING;
   arrPara[0].nParaSize := 8;

   arrPara[1].sParaName := 'TypeNum';
   arrPara[1].eParaType := Tc3_Database.E_ExpParameterType.Int32;
```
Examples

```
arrPara[1].nParaSize := 4;
arrPara[2].sParaName := 'Timestamp';
arrPara[2].eParaType := Tc3_Database.E_ExpParameterType.STRING;
arrPara[2].nParaSize := 81;
arrPara[3].sParaName := 'Tester';
arrPara[3].eParaType := Tc3_Database.E_ExpParameterType.STRING;
arrPara[3].nParaSize := 81;
arrPara[4].sParaName := 'Result';
arrPara[4].eParaType := Tc3_Database.E_ExpParameterType.STRING;
arrPara[4].nParaSize := 81;
arrPara[5].sParaName := 'Test';
arrPara[5].eParaType := Tc3_Database.E_ExpParameterType.XMLTAGName;
arrPara[5].nParaSize := 0;

sCmd := 'XPATH_ADD<ATTR>#ProductionConfig/Tests';
stTest.sTestNum := sTestNum;
stTest.nTypeNum := nTypeNum;
stTest.sTimestamp := sTimestamp;
stTest.sTester := sTester;
stTest.sResult := sResult;

CASE nState_AddEntry OF
  0:
    IF fbPLCDBCmd.Execute(
      hDBID:= 2,
      pExpression:= ADR(sCmd),
      cbExpression:= SIZEOF(sCmd),
      pData:= ADR(stTest),
      cbData:= SIZEOF(stTest),
      pParameter:= ADR(arrPara),
      cbParameter:= SIZEOF(arrPara)) THEN
      ipResultEvt := fbPLCDBCmd.ipTcResult;
      nState_AddEntry:= 100;
    END_IF
  100:
    IF _SetResultInfo(1033) THEN
      nState_AddEntry:= 0;
      AddTestEntry := TRUE;
    END_IF
END_CASE

_SetResultInfo (private method)

The I_Message message interface is evaluated by the TwinCAT EventLogger in the private _SetResultInfo method.

METHOD _SetResultInfo : BOOL
VAR_INPUT
  nLangId : INT := 1033;
END_VAR

_SetResultInfo := FALSE;

CASE nState_SetResInfo OF
  0:
    IF ipResultEvt.RequestEventText(nLangId, EventText, SIZEOF(EventText)) THEN
      nState_SetResInfo := 1;
    END_IF
  1:
    IF ipResultEvt.RequestEventClassName(nLangId, EventClassName, SIZEOF(EventClassName)) THEN
      EventId := ipResultEvt.nEventId;
      bError := (ipResultEvt.eSeverity = TcEventSeverity.Error) OR
                (ipResultEvt.eSeverity = TcEventSeverity.Critical);
      nState_SetResInfo := 0;
      _SetResultInfo := TRUE;
    END_IF
END_CASE
```

Documents about this

TF6420_Sample4_XMLProductionConfig.zip (Resources/zip/9007202749934091.zip)
7.1.2  Best practices

The tips for using the TwinCAT Database Server illustrate the benefits of the individual function blocks and applications in terms of performance, flexibility and complexity.

7.1.2.1  Writing CSV files

The TwinCAT 3 Database Server supports the CSV file format. There are different approaches, each with advantages and disadvantages, to write content to the file or read from it. Two of these approaches are explained in more detail here.

Select the ASCII database. The .csv file format can be specified under the file path. The ASCII-DB 3.0 format flag indicates the format of the ASCII/CSV file. If the format is checked, the SAX procedure is used. With this setting, write access to the file, especially with the FB_PLCDBCmdEvt function block, is also very efficient for large files. If the format is unchecked, the DOM procedure is used, which is particularly suitable for reading a file. The data is stored in a structured form in the RAM. Therefore this method is recommended for smaller files (less than 1 MB). However, this method offers some advantages due to the structured storage. The CSV file can be used as an SQL database using a stored table structure. Use the SQL Query Editor to do this. This file can be created directly via the ‘Create’ button.

![CSV File_3.0](image)

Load your configuration onto your TwinCAT Database Server target system.

**Table 2: ASCII format compatibility**

<table>
<thead>
<tr>
<th>Function block</th>
<th>Table structure</th>
<th>ASCII 3.0 format</th>
<th>Standard ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB_PLCDBWriteEvt.Write</td>
<td>standard</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FB_PLCDBWriteEvt.WriteStruct*</td>
<td>variable</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>FB_PLCDBReadEvt.Read</td>
<td>standard</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FB_:PLCDBReadEvt.ReadStruct*</td>
<td>variable</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>FB_PLCDBCmdEvt.Execute*</td>
<td>variable</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>FB_SQLCommandEvt</td>
<td>variable</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

Items marked with * are used in the following sample

**High-performance writing to the CSV file**
The most efficient way to write to a CSV file is based on the function block FB_PLCDBCmdEvt. To this end, the link to the CSV file must be set in ASCII-DB 3.0 format. The DBValueType is irrelevant here. A table structure does not have to be defined in advance.

**Sample:**

The following structure is used as an example:

```plaintext
TYPE ST_CSVDataStruct :
  STRUCT
    ID: LINT;
    Timestamp: DT;
    Name: STRING(80);
    Velocity: LREAL;
    Temperature: LREAL;
  END_STRUCT
END_TYPE
```

The function block is initialized as follows:

```plaintext
VAR
  InputData: ST_CSVDataStruct;
  fbPLCDBCmd: FB_PLCDBCmd (eNetID:= '', tTimeout := T#30S);

  sCmd : T_MaxString := '{ID};{Timestamp};{Name};{Velocity};{Temperature}';

  para : ARRAY [0..4] OF ST_ExpParameter :=
      [(eParaType:= E_ExpParameterType.Int64, nParaSize := 8, sParaName := 'ID'),
       (eParaType:= E_ExpParameterType.DateTime, nParaSize := 4, sParaName := 'Timestamp'),
       (eParaType:= E_ExpParameterType.STRING_, nParaSize := 81, sParaName := 'Name'),
       (eParaType:= E_ExpParameterType.Double64, nParaSize := 8, sParaName := 'Velocity'),
       (eParaType:= E_ExpParameterType.Double64, nParaSize := 8, sParaName := 'Temperature')];

END_VAR
```

The individual parameters are specified in curly brackets within the command. Information about the type, byte length and name is assigned via the initialization. The name is used to recognize the parameter in the command and to replace it with the value from the PLC when it is written to the file.

The call in the PLC source code of the function block consists of a call:

```plaintext
IF fbPLCDBCmd.Execute(
  hDBID:= 3,
  pExpression:= ADR(sCmd),
  cbExpression:= SIZEOF(sCmd),
  pData:= ADR(InputData),
  cbData:= SIZEOF(InputData) ,
  pParameter:= ADR(para),
  cbParameter:=SIZEOF(para))
THEN
  //Place for errorhandling or reactions;
END_IF
```

The hDBID depends on its configuration and can be taken from the database link. pData (or cbData) can be the address for the individual structure or for an array of its structure. This can lead to further performance improvements.

**Structured writing and reading of a CSV file**

Not all function blocks are possible with the ASCII format 3.0. Some functions of the TwinCAT Database Server require a preconfigured table structure. However, this cannot be stored in ASCII format 3.0. In this sample, a fixed structure is used to write and read the data with the PLCDBWriteEvt and PLCDBReadEvt function blocks in any structure.

The following structure is used as an example:
Export for the PLC under the 'Select' tab is also possible:

```
TYPE ST_CSVDataStruct :
STRUCT
   ID: LINT;
   Timestamp: DT;
   Name: STRING(80);
   Velocity: LREAL;
   Temperature: LREAL;
END_STRUCT
END_TYPE
```

The `Write/ReadStruct` methods of the respective PLC function blocks are used for any table structures:

```
VAR
   fbPLCDBWrite: FB_PLCDBWrite(sNetID:= '', tTimeout := T#30S);
   fbPLCDBRead : FB_PLCDBRead(sNetID:= '', tTimeout := T#30S);
   ColumnNames : ARRAY [0..4] OF STRING(50) := ['ID','Timestamp','Name','Velocity','Temperature'];
   Data: ST_CSVDataStruct;
   ReadData: ARRAY[0..4] OF ST_CSVDataStruct;
END_VAR

IF fbPLCDBWrite.WriteStruct(
   hDBID:= hDBID,
   sTableName:= 'CSV_Sample',
   pRecord:= ADR(Data),
   cbRecord:= SIZEOF(Data),
   pColumnNames:= ADR(ColumnNames),
   cbColumnNames:= SIZEOF(ColumnNames) )
THEN
   //Place for errorhandling or reactions
END_IF

IF fbPLCDBRead.ReadStruct(
   hDBID:= hDBID,
   sTableName:= 'CSV_Sample',
   pColumnNames:= ADR(ColumnNames),
   cbColumnNames:= SIZEOF(ColumnNames),
   sOrderByColumn:= 'ID',
   eOrderType := E_OrderType.ASC,
   nStartIndex:= 0,
   nRecordCount:= 5,
   pData:= ADR(ReadData),
   cbData:=SIZEOF(ReadData))
THEN
   //Place for errorhandling or reactions
END_IF
```

The `WriteStruct(...)` method writes the `Data` structure to the database. The structures of the PLC and the CSV file are compared based on the `ColumnNames`.

The `ReadStruct(...)` method reads a certain number (`nRecordCount`) of records from the CSV file. These may be sorted based on a selected column. The size of the `ReadData` target array should be sufficient to receive all the retrieved data.

Appendix
Sample configurations for both samples, as well as the complete code of a simple sample program, can be downloaded here: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/5778536715.zip. To illustrate the process, the program generates values and repeatedly sends them to the CSV. The settings used above were stored in a separate function block, which communicates in different ways with the two CSV formats.

Documents about this

TF6420_BestPractise_CSV.zip (Resources/zip/5778536715.zip)

7.1.2.2 Fast logging with data buffer

In order to log data in a database at millisecond intervals, the data must first be consolidated before it is transferred to the database via the TwinCAT Database Server. These data buffers can vary in size according to requirements. In the sample, 100 data samples are combined in a buffer before they are transferred with the TwinCAT Database Server. To avoid gaps during the write process, several buffers must be created in which the data samples are combined. In the sample, a total of 20 buffers are created using a 2-dimensional array.

Data sample

Definition:

```
TYPE ST_Data :
  STRUCT
    Timestamp        : LINT;
    fAM              : LREAL;
    fPeak            : LREAL;
    fPulse           : LREAL;
    fSawtooth        : LREAL;
    fSine            : LREAL;
    fSquare          : LREAL;
    fStairs          : LREAL;
    fTriangular      : LREAL;
  END_STRUCT
END_TYPE
```

Each cycle fills one element of the data buffer. In the sample this happens at 10 ms intervals. Thus a buffer contains data of a period of 1 s. If a buffer is filled with 100 elements, a further array indicates that the 100 elements can now be transferred with the function block FB_PLCDBCmdEvt. To this end, the entire buffer can be transferred to the function block. Each individual element is then transferred from the TwinCAT Database Server to the database. This sample can also be implemented with other function blocks. Note that not all function blocks support arrays.

Extract from the function block FB_Record_tbl_Signals

( "State Machine" => State: Recording)

```
  2://Recording
  bRecording := TRUE;

  //Fill buffer
  stData[nWriteBufferIndex, nWriteIndex].Timestamp := nTimestamp;
  stData[nWriteBufferIndex, nWriteIndex].fAM := fAM;
  stData[nWriteBufferIndex, nWriteIndex].fPeak := fPeak;
  stData[nWriteBufferIndex, nWriteIndex].fPulse := fPulse;
  stData[nWriteBufferIndex, nWriteIndex].fSawtooth := fSawtooth;
  stData[nWriteBufferIndex, nWriteIndex].fSine := fSine;
  stData[nWriteBufferIndex, nWriteIndex].fSquare := fSquare;
  stData[nWriteBufferIndex, nWriteIndex].fStairs := fStairs;
  stData[nWriteBufferIndex, nWriteIndex].fTriangular := fTriangular;

  //Set buffer index
  nWriteIndex := nWriteIndex + 1;
  IF nWriteIndex = 100 THEN
    nWriteIndex := 0;
    aWriteSQL[nWriteBufferIndex] := TRUE;
    nWriteBufferIndex := nWriteBufferIndex + 1;
  END_IF

  IF nWriteBufferIndex = 20 THEN
```
nWriteBufferIndex := 0;
END_IF

IF aWriteSQL[nWriteBufferIndex] THEN
  nState := 255;
  RETURN;
END_IF

// Write buffer element (100 samples) to database
IF aWriteSQL[nSQLIndex] THEN
  IF fbPLCDBCmd.Execute(nDBID, ADR(sCmd), SIZEOF(sCmd),
    ADR(stData[nSQLIndex,0]), SIZEOF(stData[nSQLIndex,0]) * 100,
    ADR(aPara), SIZEOF(aPara)) THEN
    IF fbPLCDBCmd.bError THEN
      nState := 255;
    ELSE
      nRecords := nRecords + 100;
      aWriteSQL[nSQLIndex] := FALSE;
      nSQLIndex := nSQLIndex + 1;
      IF nSQLIndex = 20 THEN
        nSQLIndex := 0;
      END_IF
    END_IF
  END_IF
END_IF

….

Appendix:

In this best practice example, a function generator block is used to generate various signals that can be logged in a database. The syntax of the INSERT command is generally valid, but has been specifically tested with an MS SQL database. The ZIP file attached below contains the complete program code in Tnzip format.

Documents about this

TF6420_BestPractise_Buffer.zip (Resources/zip/6263666699.zip)

7.1.2.3 NoSQL

This document describes the handling of NoSQL databases.

Database used: MongoDB

Database type used: DocumentDB

Data writing

Database types of type DocumentDB can store JSON documents with any structure. Therefore it is possible to map any structure of the PLC in DocumentDBs. This document can be created automatically using the FB_JSONDataType or assembled using the string blocks. Make sure that the document variable is large enough. If you want to write several documents at the same time, you can transfer them in a JSON array.

The QueryOptions[231] are defined in preparation. The collection concerned and the query type are specified for this purpose. Each query type has its own structure. The structure T_QueryOptionDocumentDB_Insert[232] is used for writing documents.

VAR
  fbNoSQLQueryBuilder_DocumentDB: FB_NoSQLQueryBuilder_DocumentDB;
  InsertQueryOptions: T_QueryOptionDocumentDB_Insert;
  sDocument : STRING(1000);  
END_VAR

InsertQueryOptions.pDocuments:= ADR(sDocument);
InsertQueryOptions.cbDocuments:= SIZEOF(sDocument);
fbNoSQLQueryBuilder_DocumentDB.eQueryType := E_DocumentDbQueryType.InsertOne;
The function block `FB_NoSQLQueryEvt` is used for writing the document into the database. The `Execute()` method writes the transferred documents to the database. This execution is asynchronous to the PLC and can take several cycles. The Boolean return value indicates when the function block has completed its process:

```plaintext
VAR
  fbNoSQLQuery: FB_NoSQLQueryEvt(sNetID := '', tTimeout := TIME#15S0MS);
  fbJsonDataType: FB_JsonReadWriteDatatype;
END_VAR
CASE eState OF
  eMyDbState.Write:
    // set the document yourself as json format (Example)
    sDocument := '{"myBool": true,
      "Name": "Some Name Value",
      "Value": 2.3,
      "Value2":3,
      "Child":{
        "Name": "Single Child",
        "Value":1,
        "myBool":true,
        "arr": [12.0,13.0,14.0,15.0],
        "myBool2": true},
      "Children": [
        {
          "Name": "Child1",
          "Value": 1,
          "myBool": true,
          "arr": [12.1,13.1,14.1,15.1],
          "myBool2": true},
        {
          "Name": "Child2",
          "Value":2,
          "myBool": true,
          "arr": [12.2,13.2,14.2,15.2],
          "myBool2": true},
        {
          "Name": "Child3",
          "Value":1,
          "myBool": true,
          "arr": [12.3,13.3,14.3,15.3],
          "myBool2": true}]
    }';
    IF fbNoSQLQuery.Execute(1, myQueryBuilder) THEN
      IF fbNoSQLQuery.bError THEN
        InfoResult := fbNoSQLQuery.ipTcResult;
      eState := eMyDbState.Error;
    ELSE
      eState := eMyDbState.Idle;
    END_IF
  END_CASE
```

The databases recognize the data type with which the individual variables are stored. However, as with MongoDB, the data type can be specified explicitly. If a timestamp is to be saved explicitly as a data type, it must be defined in the JSON document:

```plaintext
sDocument := '{"myTimestamp": ISODate("2019-02-01T14:46:06.0000000"), ...}';
```

The string can not only be formatted via the string formatting function blocks of the TwinCAT 3 libraries, but also via auxiliary function blocks for JSON documents, such as `FB_JsonReadWriteDatatype` from `Tc3_JsonXml`.

```plaintext
// set the document by JsonDataType
sTypeName := fbJsonDataType.GetDatatypeNameByAddress(SIZEOF(anyValue[1]), ADR(anyValue[1]));
sDocument := fbJsonDataType.GetJsonStringFromSymbol(sTypeName, SIZEOF(anyValue [1]), ADR(anyValue [1]));
```

### Reading data

The data schema in the document-based database can be different for each document. In contrast, the PLC follows a fixed process image. The data may not correspond to the process image.
There are two different ways of reading data in the database: the find query and the aggregation method. Both return results from the database, although aggregation offers extended options for transforming the data into an appropriate form or for performing operations, such as calculating average values directly.

The `QueryOptions` are defined in preparation. The collection concerned and the query type are specified for this purpose. Each query type has its own structure. The structure `T_QueryOptionDocumentDB_Aggregation` is used for aggregating documents.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{Place : &quot;NorthEast&quot;}</code></td>
<td>All records which have &quot;NorthEast&quot; as value of the element &quot;Place&quot;.</td>
</tr>
</tbody>
</table>
| `{myValue : { $arrayElemAt :  
{ "$WindPlantData.RotorSensor", 2 } }} ` | Returns all RotorSensor data from array element location 2 as "myValue". |
| `{RotorAvg : { $avg:
{ "$WindPlantData.RotorSensor" } }} ` | Returns the average value of the data array "RotorSensor" as "RotorAvg". |

The complete documentation of the operators is available from the respective database provider.

A reference to the returned data can now be found in the function block `FB_NoSQLResultEvt`. These can now be read as JSON documents in a string or as a structure. The data is now read directly into an array with a suitable structure. You can use the SQL Query Editor of the Database Server to directly generate a structure that matches the record. Instead of an array, it is also possible to store an address for a single structure when retrieving only one record.
CASE eState OF
  eMyDbState.ReadStruct:
    IF fbNoSqlDBResult.ReadAsStruct(0, 4, ADR(aRdStruct), SIZEOF(aRdStruct), bValidate := TRUE, ADR(fbNoSqlValidation), bDataRelease := TRUE) THEN
      IF fbNoSqlDBResult.bError THEN
        InfoResult := fbNoSqlDBResult.ipTcResult;
        eState := eMyDbState.Error;
      ELSE
        eState := eMyDbState.Idle;
      END_IF
    END_IF
  END_CASE

The TwinCAT Database Server takes into account the names of the elements in the record and the names of the variables when assigning record or structure. If these are to differ, the attribute "ElementName" can be used in the PLC:

```plaintext
TYPE ST_WindFarmData:
  STRUCT
    {attribute 'ElementName' := '_id'}
    ID: T_ObjectId_MongoDB;
    {attribute 'ElementName' := 'Timestamp'}
    LastTime: DT;
    {attribute 'ElementName' := 'WindPlantData'}
    Data: ST_WindFarmData_WindPlantData;
  END_STRUCT
END_TYPE
```

In this sample, "ElementName" specifies the name of the data in the database document. The start index and the number of records can be used to determine which records are to be returned with this call. In order to avoid possible duplications, please note that these options can already be carried out with operators at the "PipeplineStages".

**Data validation**

If there were conflicts between the record and the structure in the PLC at FB_NoSQLResult [207], they can be read out with FB_NoSQLValidationEvt [211]. Examples of conflicts are missing or surplus records, or data type problems. The method GetIssues() [212] can be used to read all conflicts as an array of strings. Surplus data that were not found in the PLC structure can be read as an array of strings in JSON format via GetRemainingData() [213]. If necessary, these can then be read out separately into the correct structure or interpreted via the TwinCAT JSON library.

```plaintext
VAR
  fbNoSqlValidation : FB_NoSQLValidationEvt(sNetID := '', tTimeout := TIME#15S0MS);
  aIssues : ARRAY[0..99] OF STRING(512);
  aRemaining : ARRAY [0..9] OF STRING(1000);
END_VAR

CASE eState OF
  eMyDbState.ValidationIssues:
    IF fbValidation.GetIssues(ADR(aIssues), SIZEOF(aIssues), FALSE) THEN
      IF fbValidation.bError THEN
        InfoResult := fbValidation.ipTcResult;
        eState := eMyDbState.Error;
      ELSE
        eState := eMyDbState.Idle;
      END_IF
    END_IF
  END_CASE
```
7.1.2.4 PostgreSQL routines

In addition to the functions (PostgreSQL routine), from PostgreSQL 11 the database also supports stored procedures to enable server-side programming. These two routine types have different properties and functions:

Table 3: Comparison of stored procedures and functions

<table>
<thead>
<tr>
<th></th>
<th>Stored Procedures</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT parameters</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Return value</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Can be used in queries</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Supports transactions</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

These properties require different interfaces with the TwinCAT Database Server. As with other supported databases, Stored Procedures are executed using the FB_SQLStoredProcedureEvt function block. Functions can be integrated in SQL commands, which are called via FB_PLCDBCmdEvt or FB_SQLCommandEvt.

PostgreSQL uses "RefCursor" for returning data sets of the routines. The TwinCAT Database Server automatically evaluates these "RefCursor" and returns the data set referenced in them. It is not possible to resolve multiple "RefCursor".

Calling a Stored Procedure

Procedures can be called via the FB_SQLStoredProcedureEvt.

Example (SQL)

SQL script for creating a procedure:

```sql
CREATE OR REPLACE PROCEDURE "public"."SP_getLastData"
(INOUT result_data refcursor)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    open result_data for SELECT * FROM "myTable_Double" LIMIT 10;
END
$BODY$;
```

If the procedure defines one (or more) "RefCursor" as output parameter(s), this (or the first) is automatically interpreted and the resulting data sets are stored in the buffer for the FB_SQLResultEvt. The data type "RefCursor" is treated like a string by the TwinCAT Database Server.

Example (TwinCAT 3 in ST)

```st
VAR
    fbSqlDatabase : FB_SQLDatabaseEvt(sNetID := '', tTimeout := T#5S);
    ParaInfo     : ST_SQLSPParameter;
END_VAR

ParaInfo.sParameterName       := '@result_data';
ParaInfo.eParameterType      := E_SPParameterType.InOut;
ParaInfo.eParameterDataType  := E_ColumnType.RefCursor; //19
ParaInfo.nParameterSize      := 81;

IF fbSQLDatabase.CreateSP('"public"."SP_getLastData"', ADR(ParaInfo), SIZEOF(ParaInfo), ADR(fbSQLStoredProcedure)) THEN
    IF fbSQLDatabase.bError THEN
        nState:=255;
    ELSE
        nState:= nState+1;
    END_IF
END_IF
```

The FB_SQLStoredProcedureEvt uses the Stored Procedure previously linked with FB_SQLDatabaseEvt.CreateSP().
**Calling a function**

Functions can be called within SQL commands.

**Example (SQL)**

SQL script for creating a function:

```sql
CREATE OR REPLACE FUNCTION "public"."F_getLastData"()
RETURNS refcursor
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE result_data refcursor;
BEGIN
  open result_data for SELECT * FROM "myTable_Double" ORDER BY "ID" DESC LIMIT 10;
  return result_data;
END;$BODY$;
```

The following SQL command is used to call the function:

```sql
SELECT "public"."F_getLastData"();
```

The call itself returns a "RefCursor". This is automatically interpreted by the TwinCAT Database Server.

**Example (TwinCAT 3 in ST)**

The `FB_SQLCommandEvt` uses the command created by `FB_SQLDatabaseEvt.CreateCmd()`. `FB_SQLResultEvt` can then be used to read the data sets.

It is advisable to use this program code in a State Machine.

**7.1.2.5  Cyclical data and time series databases**

This document describes the handling of time series and how cyclic data is stored in time series databases.
Introduction

Writing of data at regular or cyclic intervals is a common application in control technology. The data should be recorded with high time precision. Since database communication is not real-time capable, it is useful to store data that is measured regularly in a buffer. An array of the data structure can be used for this purpose. The collected data is then sent to the TwinCAT Database Server, where it can be processed without time constraints and subsequently stored in the database.

Time

Each data set that is stored in the database is assigned a timestamp. Together with the tag columns these form a unique ID. If two data sets have the same ID (same timestamp and tag values), the newer data set overwrites the old one.

Example:

<table>
<thead>
<tr>
<th>time</th>
<th>locationname (tag)</th>
<th>temperature (field)</th>
<th>windspeed (field)</th>
</tr>
</thead>
<tbody>
<tr>
<td>158167520063026200</td>
<td>158167520063027200</td>
<td>158167520063028200</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Verl</td>
<td>1.15</td>
<td>6.3</td>
</tr>
<tr>
<td>2</td>
<td>Verl</td>
<td>1.03</td>
<td>5.2</td>
</tr>
<tr>
<td>3</td>
<td>Verl</td>
<td>0.98</td>
<td>2.8</td>
</tr>
<tr>
<td>4</td>
<td>Hamburg</td>
<td>1.42</td>
<td>1.49</td>
</tr>
<tr>
<td>5</td>
<td>Hamburg</td>
<td>1.56</td>
<td>8.9</td>
</tr>
</tbody>
</table>

Data set no. 4 is overwritten by a new data set, since the ID is identical.

The timestamp in the database is saved by default as UNIX epoch time. With the exception of the user-created insert commands, the timestamps are received and converted as TwinCAT time (number of 100 ns steps since January 1, 1601) in the function blocks of the TwinCAT 3 Database Server. The times are not converted for insert commands.

Database configuration

InfluxDB should be selected in the database configuration. Insert the connection parameters for the required database. If no database is available yet, you can create the database by clicking the "Create" button. Pay attention to your firewall settings or port approvals. A table does not have to be created, since it is automatically created during the first InfluxDB access. InfluxDB does not have a fixed table schema. Columns can also be extended or added later.

Writing cyclic data

This example shows how symbols can be written from the PLC into a time series database with minimal effort.

Declaration:

```plaintext
// databuffer for 1 second with 10 ms time delta
windTurbineData: ARRAY[1..100] OF WindTurbineData;
```
Declaring the data source structure:

In this structure the attributes "TagName" and "FieldName" are used to declare the data fields as tags or fields. By default they are declared as fields. These attributes can also be used if you want the column name in the table to differ from the symbol name in the PLC.

To capture unset data during data analysis, default values outside the value range can be used to detect such data during the analysis.

```plaintext
TYPE WindTurbineData:
  STRUCT
    {attribute 'TagName' := 'ID'}
    WindTurbineID : STRING(255);
    {attribute 'FieldName' := 'Power'}
    Power : LREAL := -1; // [0) [kW]
    {attribute 'FieldName' := 'Wind Speed'}
    WindSpeed : LREAL := -1; // [0) [m/s]
    {attribute 'FieldName' := 'Wind Direction'}
    WindDirection : LREAL := -1; // [0,360)[°]
  END_STRUCT
END_TYPE
```

(\textit{WindTurbineData.tcDUT})

Declaration of the ENUM for the State-Machine:

```plaintext
TYPE E_DbLogState:
  {id, init, writing, error}
END_TYPE
```

Generating sample data:

```plaintext
FOR i := 1 TO 100 BY 1 DO
  rand();
  nrand := rand.Num;
  windTurbineData[i].WindDirection := 240.328 + nrand;
  windTurbineData[i].WindSpeed := 7.3292 + nrand;
  windTurbineData[i].Power := 1133.1975 + nrand;
END_FOR
```

Preparing the call:

```plaintext
CASE State OF
  E_DbLogState.idle:
    IF bWriting THEN
      bWriting := FALSE;
      State := E_DbLogState.init;
    END_IF
END_CASE
```
In this case, the array 'windTurbineData' is written into the 'WindMeasurement' table of the database. Therefore the data is read directly from the process image. The data type is specified to read the addresses in the memory. The time of the data sets is automatically generated by specifying the start time and the time interval. The data must be stored correctly in the array. For example, one data set in the array can be used per PLC cycle. It is useful to create a PLC task for this process. In this example the cycle time is 10 ms.

```plaintext
E_DbLogState.init:

fbNoSQLQueryBuilder_TimeSeriesDB.pQueryOptions := ADR(QueryOption_TSDB_Insert);
fNoSQLQueryBuilder_TimeSeriesDB.cbQueryOptions := SIZEOF(QueryOption_TSDB_Insert);
QueryOption_TSDB_Insert.sTableName := 'WindMeasurement';
QueryOption_TSDB_Insert.sDataType := 'WindTurbineData';
QueryOption_TSDB_Insert.pSymbol := ADR(windTurbineData);
QueryOption_TSDB_Insert.cbSymbol := SIZEOF(windTurbineData);
QueryOption_TSDB_Insert.nDataCount := 100;
QueryOption_TSDB_Insert.nStartTimestamp := F_GetSystemTime();
QueryOption_TSDB_Insert.nCycleTime := 10000; // (in 100 ns)
State := E_DbLogState.writing;
```

**Writing the data:**

This call writes the data to the configured database with the corresponding database ID. This can take several cycles, since it is an asynchronous process. If necessary, several storage arrays must be used to ensure that the data is recorded seamlessly without gaps.

```plaintext
E_DbLogState.writing:

IF fbNoSqlQueryEvt.Execute(dbid, fbNoSQLQueryBuilder_TimeSeriesDB) THEN
  IF fbNoSqlQueryEvt.bError THEN
    TcResult := fbNoSqlQueryEvt.ipTcResult;
    State := E_DbLogState.error;
  ELSE
    State := E_DbLogState.idle;
  END_IF
END_IF
```

**Error handling:**

Use the Tc3 Eventlogger for error handling

```plaintext
E_DbLogState.error:

IF TcResult.RequestEventText(1033, sErrorMessage, SIZEOF(sErrorMessage)) THEN
  TcResult.Send(F_GetSystemTime());
  State := E_DbLogState.idle;
  bError := TRUE;
END_IF
END_CASE
```

### 7.2 Tc2_Database

All sample applications for the TwinCAT Database Server were consolidated in a solution. The solution can be downloaded here from a central location: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip

In addition to the tszip file for the TwinCAT 3 solution, the zip file contains all the required file-based databases. If the folder "Samples" from the zip file is located in the default installation folder: C:\TwinCAT\Functions\TF6420-Database-Server\Win32, the paths in the Database Server configuration do not have to be edited further. The samples with non-file-based databases, such as MS SQL, have to be individual adapted with the configurator.

The individual samples are documented in detail on separate pages:
### Examples

<table>
<thead>
<tr>
<th>SP project name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create_DB_Sample [358]</td>
<td>Creating a database connection and a table from the PLC</td>
</tr>
<tr>
<td>Cyclic_RdWrt_Sample [361]</td>
<td>Cyclic logging/writing to/from a database</td>
</tr>
<tr>
<td>Write_DB_Sample [362]</td>
<td>Writing of variables into a database with a simple PLC function block without SQL command</td>
</tr>
<tr>
<td>SQL_InsertSelect_Sample [366]</td>
<td>Sample with function block FB_DBRecordInsert/FB_DBRecordArraySelect</td>
</tr>
<tr>
<td>StoredProcedures_Sample [368]</td>
<td>Stored procedures with FB_DBStoredProceduresRecordArray</td>
</tr>
<tr>
<td>XML_DB_Sample [371]</td>
<td>Using XML files as database</td>
</tr>
<tr>
<td>XML_XPath_Sample [376]</td>
<td>XML XPath sample without schema</td>
</tr>
<tr>
<td>XML_XPath_Schema_Sample [379]</td>
<td>XML XPath sample with XML Schema, comparable with TwinCAT XML Server &quot;Read&quot;</td>
</tr>
</tbody>
</table>

### 7.2.1 Creating an MS Access database

This example illustrates the creation of a database from the PLC. In addition, a table is added, and the database that has been generated is declared in the XML configuration file.

**Download:** https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip

![XML - Configuration File](image)

**StartUp := Manual**

**PLC Library**

- FB_DBCreate
- FB_DBConectionAdd
- FB_DBTableCreate

Creates a new Database with a Table and adds the XML Database declaration to the XML-Configuration file.
Database type used | MS Access
---|---
Compatible database types | MS SQL, MS Compact SQL, MS Access, XML
Function blocks used | FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate
Libraries to be integrated | Tc2_Database, Tc2_System, Tc2_Standard
Download file list | TcDBSrv_InfoSysSamples.tszip

A table with the name "myTable", which has the following structure, is added to the generated database:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Bigint</td>
<td>IDENTITY(1,1)</td>
</tr>
<tr>
<td>Timestamp</td>
<td>datetime</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Ntext</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
<td></td>
</tr>
</tbody>
</table>

This table structure is generated with the following array:

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

### Variable Declaration

```pascal
PROGRAM MAIN
VAR
  R_TRIG1          : R_TRIG;
  bSTART           : BOOL;
  FB_FileDelete1   : FB_FileDelete;
  FB_DBCreate1     : FB_DBCreate;
  FB_DBConnectionAdd1: FB_DBConnectionAdd;
  FB_DBTableCreate1: FB_DBTableCreate;
  bBusy_Delete     : BOOL;
  bBusy_CreateDB   : BOOL;
  bBusy_ConnAdd    : BOOL;
  bBusy_CreateTable: BOOL;
  bErr             : BOOL;
  nErrid           : UDINT;
  nDBid            : UDINT;
  arrTablestrc     : ARRAY [0..3] OF ST_DBColumnCfg :=
[(sColumnName:='ID',sColumnProperty:='IDENTITY(1,1)',eColumnType:=EDBCOLUMN_BIGINT),
 (sColumnName:='Timestamp',eColumnType:=EDBCOLUMN_DATETIME),
 (sColumnName:='Name',eColumnType:=EDBCOLUMN_NTEXT),
 (sColumnName:='Value',eColumnType:=EDBCOLUMN_FLOAT)];
  nState:BYTE := 0;
END_VAR
```

### PLC program

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

IF NOT bBusy_Delete THEN
  nState := 2;
END_IF

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

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

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

IF NOT bBusy_ConnAdd AND NOT bErr THEN
  nState := 4;
END_IF

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

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

END_CASE

In order to use this sample, you only need to transfer the NetID of the ADS device (on which the TwinCAT Database Server is installed) to the sNetID input.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
7.2.2 Starting / stopping, cyclic logging

This sample illustrates the starting and stopping of cyclic logging from the PLC.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip

In this sample the cyclic log function is started or stopped by toggling the bStartStop variable. The cyclic log process begins in response to a positive edge at the bExecute input. A negative edge will end the process again.

Variable declaration (PRG data types)

```plaintext
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_MSTRUCT: ST MyStruct;
  DBSrv_DT_ARRAY : ARRAY [0..19] OF UDINT;
```
Examples

```
DBSrv_DT_WORD   : WORD;
DBSrv_DT_DWORD  : DWORD;
END_VAR

ST_MyStruct structure

TYPE ST_MyStruct :
STRUCT
  iValue1 : INT;
  iValue2 : UINT;
  iValue3 : BOOL;
  iValue4 : REAL;
END_STRUCT
END_TYPE

Variable Declaration

PROGRAM MAIN
VAR
  fbDBCyclicRdWrt1: FB_DBCyclicRdWrt;
  bCyclic         : BOOL := TRUE;
  bBusy_Cyclic   : BOOL;
  bErr            : BOOL;
  nErrID          : UDINT;
  sSQLState      : ST_DBSQLError;
END_VAR

PLC program

DataTypes;

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

In order to use this sample, you only need to transfer the NetID of the ADS device (on which the TwinCAT Database Server is installed) to the sNetID input. When you run the program and set the bCyclic variable to TRUE, all the variables that are declared in the symbol group of the XML configuration file are logged.

Information

TwinCAT Database Server

All Microsoft SQL Compact databases, which are declared in the XML configuration file, must exist. They are not generated automatically.

The declared ASCII files, on the other hand, are generated automatically if they do not exist.

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

7.2.3 Logging of a PLC variable with FB_DBWrite

This sample illustrates logging of a PLC variables from the PLC in a database and the operating principle of the individual write modes.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip
In order to be able to use this sample, you have to adapt the server name and the authentication in the XML configuration file (CurrentConfigDataBase.xml). Ensure that no "TestDB" database is present before executing the SQLQuery.sql script.

**Sample configuration:**

The variable "eWriteMode" can be used to set the write mode for logging. The write operation can then be started with a positive edge at the variable "bSTART".

**Table assignment:**

- **ADS_TO_DB_APPEND** => eWriteAppend -> "tbl_Append"
- **ADS_TO_DB_UPDATE** => eWriteUpdate -> "tbl_Update"
- **ADS_TO_DB_RINGBUFFER** => eWriteRingBuffer -> "tbl_RingBuffer"
### Table structure used

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Null permitted</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Bigint</td>
<td>no</td>
<td>IDENTITY(1,1)</td>
</tr>
<tr>
<td>Timestamp</td>
<td>datetime</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Ntext</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

### Variable Declaration

```plaintext
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
    eWriteAppend :
        FB_DBWrite_Append(  
            sNetID         := ,  
            hDBID          := 1,  
            hAdsID         := 1,  
            sVarName       := 'MAIN.lrTestValueVar',  
            nVarSize       := ,  
            sVarType       := ,  
            sDBVarName     := 'lrTestValueVar',  
            eDBWriteMode   := eDBWriteMode_Append,  
            tRingBufferTime := ,  
            nRingBufferCount:= ,
            bExecute       := TRUE,
            tTimeout       := T#15s,
            bBusy          => bBusy,
            bError         => bErr,
            nErrID         => bErrid,
```
sSQLState => stSqlstate);

IF NOT bBusy THEN
  FB_DBWrite_Append(bExecute := FALSE);
eState := eIdle;
END_IF

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

IF NOT bBusy THEN
  FB_DBWrite_Update(bExecute := FALSE);
eState := eIdle;
END_IF

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

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

END_CASE

---

**TwinCAT Database Server**

All Microsoft SQL Compact databases, which are declared in the XML configuration file, must exist. They are not generated automatically.

The declared ASCII files, on the other hand, are generated automatically if they do not exist.

---

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
7.2.4 Example with the FB_DBRecordInsert and FB_DBRecordSelect function blocks

This example illustrates logging of several values in a database from the PLC with the function block FB_DBRecordInsert. In this example, several PLC variables are logged in a single record. In addition, the function block FB_DBRecordSelect can be used to read a record from this database.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip

<table>
<thead>
<tr>
<th>Database type used</th>
<th>MS Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatible database types</td>
<td>MS SQL, MS Compact SQL, MS Access, MySQL, PostgreSQL, DB2, Oracle, InterBase/Firebird, XML</td>
</tr>
<tr>
<td>Function blocks used</td>
<td>FB_DBRecordInsert, FB_DBRecordSelect</td>
</tr>
<tr>
<td>Libraries to be integrated</td>
<td>&quot;Tc2_Database&quot;, &quot;Tc2_System&quot;, &quot;Tc2_Standard&quot;, &quot;Tc2_Utilities&quot;</td>
</tr>
<tr>
<td>Download file list</td>
<td>TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml, TestDB_Access.mdb</td>
</tr>
</tbody>
</table>

The following table structure is used for writing:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>datetime</td>
</tr>
<tr>
<td>PLC_TestValue1</td>
<td>float</td>
</tr>
<tr>
<td>PLC_TestValue2</td>
<td>float</td>
</tr>
<tr>
<td>PLC_TestValue3</td>
<td>float</td>
</tr>
<tr>
<td>PLC_TestValue4</td>
<td>String</td>
</tr>
</tbody>
</table>

Variable Declaration

```pascal
(* Declaration *) PROGRAM MAIN
VAR
  eState : E_SQLStatement;
  NT_GetTime1 : NT_GetTime;
  bTimestart : BOOL;
```
TF6420  Version: 1.11  367

```plaintext
Example:

```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 := TestValue2,
      arg3 := TestValue3,
      arg4 := TestValue4);
  END_CASE
```
Examples

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,
  eSQLState => 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 => bErrSelect,
  nErrID => nErrIDSelect,
  eSQLState => stSQLStateSelect,
  nRecords => nRecordCount);
END_CASE

To use this sample, you have to declare the Access database "Sample7.mdb" in the XML configuration file. A record with the four PLC values and the timestamp is created in the database by generating a positive edge at the variable "bStartstopInsert".

![Database Table](image)

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

7.2.5 Stored procedures with FB_DBStoredProceduresRecordArray

The function block FB_DBStoredProceduresRecordArray can be used to declare parameters as INPUT, OUTPUT or INOUT and transfer them to the stored procedures. In this way complex SQL commands can be preprogrammed in the database server and then triggered by the TwinCAT Database Server. In contrast to the function block FB_DBStoredProceduresRecordReturn, this function block can be used to return several records with a single call.
Examples

Database type used: MS SQL (MS SQL Server 2008)
Compatible database types: MS SQL, MySQL, Oracle
Function blocks used: FB_DB Stored Procedures Record Array
Libraries to be integrated: Tc2_Database, Tc2_System, Tc2_Base, Tc2_Utilities
Download file list: TcDBSrv_InfoSysSamples.tszip, CurrentConfigDataBase.xml

The following sample illustrates the call in a simple stored procedure with an input parameter and return record. The procedure was created on a Microsoft SQL Server 2008.

**Code der Stored Procedure SP_GetAddressByCustomerID**

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

**Variable declaration in the PLC**

```plaintext
PROGRAM MAIN
VAR
    R_TRIG1 : R_TRIG;
    bREAD : BOOL := FALSE;
    nState : BYTE;
```
Record structure in the PLC (ST_Record)

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

PLC program

R_TRIG1(CLK:=bREAD);
IF R_TRIG1.Q AND NOT bBusy THEN
  nState := 1;
END_IF
CASE nState OF
  0:
  ;
  1:("Init of the parameters")
    arrParaList[0].sParameterName := '@Customer_ID';
    arrParaList[0].eParameterDataType:= eDBColumn_Integer;
    arrParaList[0].eParameterType := eDBParameter_Input;
    arrParaList[0].cbParameterValue := SIZEOF(nCustomerID);
    arrParaList[0].pParameterValue := ADR(nCustomerID);
    nState := 2;
  2:("Start the stored procedure "SP_GetCustomerPosition"")
    FB_DBStoredProceduresRecordArray1(sNetID:= ,
      hDBID:= 1,
      sProcedureName := 'SP_GetCustomerPositions',
      pParameterList := ADR(arrParaList),
      nStartIndex := nRecordStartIndex,
      nRecordCount := 25,
      cbRecordArraySize:= SIZEOF(stRecordArr),
      pDestAddr := ADR(stRecordArr),
      bExecute := TRUE,
      tTimeout := T#15s,
      bBusy => bBusy,
      bErr => bErr,
      nErrID => nErrid,
      stSqlstate => stxDBSQLLError,
      nRecords => nRecs);
    IF NOT bBusy THEN
      FB_DBStoredProceduresRecordReturn1(bExecute:= FALSE);
      nState := 0;
    END_IF
END_CASE
Visualization

![Visualization Image]

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

7.2.6 Using XML as database

The TwinCAT Database Server provides the ability to use an XML file as a database. Apart from the "Stored Procedure" functions, the XML database type supports all known function blocks for reading and writing in a database. Even SQL commands that can be issued with the function blocks FB_DBRecordInsert or FB_DBRecordSelect are interpreted by the TwinCAT Database Server and applied to the XML file.

This sample demonstrates how an XML database is created, filled with the function block FB_DBWrite and subsequently read with an SQL SELECT command and the function block FB_DBRecordSelect.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip
TF6420

Version: 1.11

Database type used
XML

Compatible database types
MS SQL, MS Compact SQL, MS Access, XML

Function blocks used
FB_DBCreate, FB_DBConnectionAdd, FB_DBTableCreate, FB_DBWrite, FB_DBRecordSelect

Libraries to be integrated
"Tc2_Database", "Tc2_System", "Tc2_Standard", "Tc2_Utilities"

Download file list
TcDBSrv_InfoSysSamples.tszip

 MAIN program
PROGRAM MAIN
VAR
  nState : BYTE := 0;
  R_TRIG1 : R_TRIG;
  bSTART : BOOL;
  nCounter : INT;
  FB_FileDelete1 : FB_FileDelete;
  FB_DBCreate1 : FB_DBCreate;
  FB_DBConnectionAdd1 : FB_DBConnectionAdd;
  FB_DBTableCreate1 : FB_DBTableCreate;
FB_DBWrite1 : FB_DBWrite;
FB_DBRecordSelect1 : FB_DBRecordSelect;

bBusy_Delete : BOOL;
bBusy_CreateDB : BOOL;
bBusy_ConnAdd : BOOL;
bBusy_CreateTable : BOOL;
bBusy_WriteDB : BOOL;
bBusy_SelectRecord : BOOL;
bErr : BOOL;
nErrId : UDINT;
stSQLState : ST_DBSQLError;
nRecs : UDINT;
nDBid : UDINT;

arrTablestrc : ARRAY [0..3] OF ST_DBColumnCfg :=
[(sColumnName:='ID',sColumnProperty:='IDENTITY(1,1)',eColumnType:=EDBCOLUMN_BIGINT),
 (sColumnName:='Timestamp',eColumnType:=EDBCOLUMN_DATETIME),
 (sColumnName:='Name',sColumnProperty:='80',eColumnType:=EDBCOLUMN_NTEXT),
 (sColumnName:='Value',eColumnType:=EDBCOLUMN_FLOAT)];
rTestValue := LREAL := 1234.56789;
stRecord : ST_Record;
END_VAR

CASE nState OF
  0:
    (*To start this sample you have to set a rising edge to the variable bSTART*)
    R_TRIG1(CLK:=bSTART);
    IF R_TRIG1.Q THEN
      nState := 1;
      FB_FileDelete1(bExecute:=FALSE);
      FB_DBCreate1(bExecute:=FALSE);
      FB_DBConnectionAdd1(bExecute:=FALSE);
      FB_DBTableCreate1(bExecute:=FALSE);
      FB_DBWrite1(bExecute:=FALSE);
      FB_DBRecordSelect1(bExecute:=FALSE);
      bSTART := FALSE;
      nCounter:= 0;
    END_IF
  1:
    (*It isn't possible to overwrite an existing database file.
     If the database file exist the FB_FileDelete block will delete the file*)
    IF NOT bBusy_Delete THEN
      nState := 10;
    END_IF
  10:
    (*It isn't possible to overwrite an existing database file.
     If the database file exist the FB_FileDelete block will delete the file*)
    IF NOT bBusy_Delete THEN
      nState := 2;
    END_IF

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

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

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

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

IF NOT bBusy_WriteDB AND NOT bErr THEN
  FB_DBWrite1(bExecute := FALSE);
  nCounter := nCounter + 1;
ENDIFnCounter = 5 THEN
  nState := 6;
Examples

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,
chRecordSize := SIZEOF(stRecord),
pDestAddr := ADR(stRecord),
bExecute := TRUE,
tTimeout := 3s,
bBusy => bBusy_SelectRecord,
bError => bErr,
nErrID => nErrid,
sSQLState => stSQLState,
nRecords => nRecs);
IF NOT bBusy_SelectRecord AND NOT bErr THEN
nState := 0;
END_IF
END_CASE

The process is started with a positive edge at the toggle variable bSTART.

The following files are created:

XMLTestDB.xml (XML database file)

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

XMLTestDB.xsd (XML Schema)

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="XMLTestDB">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="myTable" type="myTable_Type"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
### 7.2.7 XPath sample to illustrate the different SELECT types

The function block FB_DBRecordArraySelect/FB_DBRecordSelect can be used to issue XPath commands and read XML tags from any XML file. This sample illustrates reading of different entries from XML files via the TwinCAT Database Server. Individual tags, subtags and reading of attributes is supported, and these are displayed.

**Download:** [https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip](https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip)

---

**Database type used** | XML
---|---
**Compatible database types** | XML
**Function blocks used** | FB_DBRecordSelect
**Libraries to be integrated** | Tc2_Database, Tc2_System, Tc2_Standard, Tc2_Utilities
**Download file list** | TcDBSrv_InfoSysSamples.tszip

---

**Sample XML file (XMLFactoryXY.xml)**

```
<?xml version="1.0" encoding="utf-8" ?>
<Factory_XY>
  <Name>Sample Factory XY</Name>
  <Factory_Info>
    <Street>Samplestreet 25</Street>
    <City>33415 Verl</City>
    <Country>Germany</Country>
    <Office_Count>1</Office_Count>
    <Employee_Count>6</Employee_Count>
    <Manager>Max Mustermann</Manager>
  </Factory_Info>
</Factory_XY>
```
<Employees>
<Employee id="10001" name="Julia Kingston" department="Development" position="Worker"
 hired="2001-08-01" />
<Employee id="10002" name="Jens Marx" department="Import" position="Worker" hired="2003-08-01" />
<Employee id="10003" name="Justus Kaiser" department="Export" position="Worker" hired="2003-08-01" />
<Employee id="10004" name="Marc Klein" department="Production" position="Worker" hired="2005-08-01" />
<Employee id="10005" name="Matt Bloomberg" department="Production" position="Worker"
 hired="2005-08-01" />
<Employee id="10006" name="Frida Hundt" department="Production" position="Worker"
 hired="2010-08-01" />
</Employees>

ST_FactoryInfo structure

TYPE ST_FactoryInfo :
   STRUCT
      sStreet      : T_MaxString;
      sCity        : T_MaxString;
      sCountry     : T_MaxString;
      sOffice_Count : T_MaxString;
      sEmploye_Count: T_MaxString;
      sManager     : T_MaxString;
   END_STRUCT
END_TYPE

ST_Employee structure

TYPE ST_Employee :
   STRUCT
      sID          : T_MaxString;
      sName        : T_MaxString;
      sDepartment  : T_MaxString;
      sPosition    : T_MaxString;
      sHired       : T_MaxString;
   END_STRUCT
END_TYPE

MAIN program

PROGRAM MAIN
VAR
   bSTART    : BOOL;
   R_TRIG1   : R_TRIG;
   nState    : INT;
   sXPath    : T_MaxString;
   fbDBRecordArraySelect : FB_DBRecordArraySelect;
   bBusy_ReadFactoryName : BOOL;
   bError_ReadFactoryName : BOOL;
   nErrID_ReadFactoryName : UDINT;
   bBusy_ReadFactoryInfo : BOOL;
   bError_ReadFactoryInfo : BOOL;
   nErrID_ReadFactoryInfo : UDINT;
   bBusy_ReadEmployee : BOOL;
   bError_ReadEmployee : BOOL;
   nErrID_ReadEmployee : UDINT;
   stSQLState     : ST_DBSQLError;
   sFactoryName   : T_MaxString;
   stFactoryInfo  : ST_FactoryInfo;
   aEmployees     : ARRAY [1..10] OF ST_Employee;
END_VAR

R_TRIG1(CLK:=bSTART);
IF R_TRIG1.Q THEN
   bSTART:=FALSE;
   fbDBRecordArraySelect(bExecute:=FALSE);
   nState:=1;
END_IF
CASE nState OF
  0://IDLE
    ;
  1://Read Factory Name
    sXPath := 'XPATH#Factory_XY/Name';
    fbDBRecordArraySelect(
      sNetID := ,
      hDBID := 7,
      pCmdAddr := ADR(sXPath),
      cbCmdSize := SIZEOF(sXPath),
      nStartIndex := 0,
      nRecordCount := 1,
      pDestAddr := ADR(sFactoryName),
      cbRecordArraySize := SIZEOF(sFactoryName),
      bExecute := TRUE,
      tTimeout := T#15S,
      bBusy => bBusy_ReadFactoryName,
      bError => bError_ReadFactoryName,
      nErrID => nErrID_ReadFactoryName,
      sSQLState => stSQLState,
      nRecords => );
    IF NOT bBusy_ReadFactoryName THEN
      fbDBRecordArraySelect(bExecute:=FALSE);
      IF NOT bError_ReadFactoryName THEN
        nState := 2;
      ELSE
        nState := 255;
      END_IF;
    END_IF
  2://Read Factory Info
    sXPath := 'XPATH#Factory_XY/Factory_Info';
    fbDBRecordArraySelect(
      sNetID := ,
      hDBID := 7,
      pCmdAddr := ADR(sXPath),
      cbCmdSize := SIZEOF(sXPath),
      nStartIndex := 0,
      nRecordCount := 1,
      pDestAddr := ADR(stFactoryInfo),
      cbRecordArraySize := SIZEOF(stFactoryInfo),
      bExecute := TRUE,
      tTimeout := T#15S,
      bBusy => bBusy_ReadFactoryInfo,
      bError => bError_ReadFactoryInfo,
      nErrID => nErrID_ReadFactoryInfo,
      sSQLState => stSQLState,
      nRecords => );
    IF NOT bBusy_ReadFactoryInfo THEN
      fbDBRecordArraySelect(bExecute:=FALSE);
      IF NOT bError_ReadFactoryInfo THEN
        nState := 3;
      ELSE
        nState := 255;
      END_IF;
    END_IF
  3://Read Employees
    sXPath := 'XPATH#Factory_XY/Employees/Employee';
    fbDBRecordArraySelect(
      sNetID := ,
      hDBID := 7,
      pCmdAddr := ADR(sXPath),
      cbCmdSize := SIZEOF(sXPath),
      nStartIndex := 0,
      nRecordCount := 10,
      pDestAddr := ADR(aEmployees),
      cbRecordArraySize := SIZEOF(aEmployees),
      bExecute := TRUE,
      tTimeout := T#15S,
      bBusy => bBusy_ReadEmployee,
      bError => bError_ReadEmployee,
      nErrID => nErrID_ReadEmployee,
      sSQLState => stSQLState,
      nRecords => );
    IF NOT bBusy_ReadEmployee THEN
      fbDBRecordArraySelect(bExecute:=FALSE);
      IF NOT bError_ReadEmployee THEN
        nState := 0;
      ELSE
        nState := 255;
      END_IF;
    END_IF
END_IF

END
nState := 255;

END_IF

255://Error State

END_CASE

A positive edge at the variable "bStart" triggers issuing of the XPath commands and reading of the individual elements from the XML file. The results will then be in the variables "sFactoryName", "stFactoryInfo" and "aEmployees".

Requirements

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>

7.2.8 XPath sample with XML schema

The function blocks FB_DBRecordSelect or FB_DBRecordArraySelect can be used to issue XPath commands and to read XML tags, XML subtags or XML attributes from any XML file. If a suitable XML schema exists for the XML file to be read, the content of the tags or attributes is converted to the corresponding data types, as defined in the schema.

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

In this sample, FB_DBRecordArraySelect is used to read two different subtags from an XML file with corresponding XML schema.

Download: https://infosys.beckhoff.com/content/1033/TF6420_Tc3_Database_Server/Resources/zip/3494041099.zip
<table>
<thead>
<tr>
<th>Database type used</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compatible database types</td>
<td>XML</td>
</tr>
<tr>
<td>Function blocks used</td>
<td>FB_DBRecordSelect</td>
</tr>
<tr>
<td>Libraries to be integrated</td>
<td>Tc2_Database, Tc2_System, Tc2_Standard, Tc2_Utilities</td>
</tr>
<tr>
<td>Download file list</td>
<td>TcDBSrv_InfoSysSamples.tszip, CurrentConfigDatabase.xml, PLC_Structs.xml, PLC_Structs.xsd</td>
</tr>
</tbody>
</table>

**Sample XML file (PLC_Structs.xml)**

```xml
<?xml version = "1.0" encoding="utf-8" ?>
<Beckhoff_PLC>
  <PLC_Structs>
    <PLC_Struct Name="ST_TestStruct">
      <Struct Instance="1">
        <nINT64>123456789</nINT64>
        <nUINT16>1234</nUINT16>
        <rREAL64>1234.5678</rREAL64>
        <sSTRING>This is instance one of ST_TestStruct</sSTRING>
        <bBOOL>true</bBOOL>
        <nINT32>-100</nINT32>
      </Struct>
      <Struct Instance="2">
        <nINT64>234567890</nINT64>
        <nUINT16>2345</nUINT16>
        <rREAL64>234.56789</rREAL64>
        <sSTRING>This is instance two of ST_TestStruct</sSTRING>
        <bBOOL>false</bBOOL>
        <nINT32>-50</nINT32>
      </Struct>
      <Struct Instance="3">
        <nINT64>345678901</nINT64>
        <nUINT16>3456</nUINT16>
        <rREAL64>3456.78901</rREAL64>
        <sSTRING>This is instance three of ST_TestStruct</sSTRING>
        <bBOOL>true</bBOOL>
        <nINT32>-150</nINT32>
      </Struct>
    </PLC_Struct>
    <PLC_Struct Name="ST_TestStruct2">
      <Struct2 Instance="1">
        <sSTRING>This is instance one of ST_TestStruct2</sSTRING>
        <bBOOL>false</bBOOL>
        <nINT32>-88</nINT32>
      </Struct2>
      <Struct2 Instance="2">
        <sSTRING>This is instance two of ST_TestStruct2</sSTRING>
        <bBOOL>true</bBOOL>
        <nINT32>-9</nINT32>
      </Struct2>
    </PLC_Struct>
  </PLC_Structs>
</Beckhoff_PLC>
```

**Corresponding XML schema (PLC_Structs.xsd)**

```xml
<?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 name = "nINT64" type="xs:long" />
                    <xs:element name = "nUINT16" type="xs:unsignedShort" />
                    <xs:element name = "rREAL64" type="xs:double" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:element name = "sSTRING" type="xs:string" />
<xs:element name = "bBOOL" type="xs:boolean" />
<xs:element name = "nINT32" type="xs:int" />

<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:attribute name = "Name" type="xs:string" use="required" />
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

Structure1 ST_TestStruct

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

Structure2 ST_TestStruct2

TYPE ST_TestStruct2 :
STRUCT
   sSTRING: T_MaxString;
bBOOL : BOOL;
nINT32 : DINT;
END_STRUCT
END_TYPE

MAIN program

PROGRAM MAIN
VAR
   nState : BYTE;
   R_TRIG1 : R_TRIG;
bStartStop : BOOL;
sCmd : T_MaxString;
FB_DBRecordArraySelect1: FB_DBRecordArraySelect;
arrTestStruct : ARRAY [0..3] OF ST_TestStruct;
arrTestStruct2 : ARRAY [0..3] OF ST_TestStruct2;
bBusy : BOOL;
bError : BOOL;
nErrID : UDINT;
stSQLState : ST_DBSQLLError;
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*)
;
sCmd:='XPATH<SUBTAG>#/Beckhoff_PLC/PLC_Structs/PLC_Struct[@Name=$"ST_TestStruct$']/Struct';
FB_DBRecordArraySelect1(
sNetID := ,
hDBID := 1,
cbCmdSize := SIZEOF(sCmd),
pCmdAddr := ADR(sCmd),
nStartIndex := 0,
nRecordCount := 4,
cbRecordArraySize := SIZEOF(arrTestStruct),
pDestAddr := ADR(arrTestStruct),
bExecute := TRUE,
tTimeout := T#15s,
bBusy => bBusy,
bError => bError,
nErrID => nErrID,
sSQLState => stSQLState,
nRecords => nRecs1);

IF NOT bBusy THEN
FB_DBRecordArraySelect1(bExecute:=FALSE);
IF NOT bError THEN
nState := 2;
ELSE
nState := 255;
END_IFEND_IF
2:
sCmd:='XPATH<SUBTAG>#Beckhoff_PLC/PLC_Structs/PLC_Struct[@Name=$"ST_TestStruct2$']/Struct2';
FB_DBRecordArraySelect1(
sNetID := ,
hDBID := 1,
cbCmdSize := SIZEOF(sCmd),
pCmdAddr := ADR(sCmd),
nStartIndex := 0,
nRecordCount := 4,
cbRecordArraySize := SIZEOF(arrTestStruct2),
pDestAddr := ADR(arrTestStruct2),
bExecute := TRUE,
tTimeout := T#15s,
bBusy => bBusy,
bError => bError,
nErrID => nErrID,
sSQLState => stSQLState,
nRecords => nRecs2);

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

Reading is started with a positive edge at the toggle variable "bStartStop".
Examples

**Requirements**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to be linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT v3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Database</td>
</tr>
</tbody>
</table>
8 Appendix

8.1 Error codes

8.1.1 Tc3_Database

8.1.1.1 PLC return values

The error output of all PLC blocks of the Tc3_Database.compiled library takes place via the I_TcResultEvent interfaces from the Tc3_Eventlogger.compiled library. This new interface structure enables a more detailed description of events, as well as classification.

Interface I_TcMessage

- nEventId: UDINT
- EventClass: GUID
- eSeverity: TcEventSeverity
- ipSourceInfo: I_TcSourceInfo

nEventID: specific event code

EventClass: GUID

EventClassName: The corresponding event class name can be read using the RequestEventClassName method

eSeverity: events classification: from "info" to "critical error"

ipSourceInfo: path to the event location.

Text: description of the event in plain text can be read with the RequestEventText method

The following event classes can occur:

- **TC3 ADS Error**
  ADS errors that may occur during the communication with the TwinCAT Database Server.

- **TC3 Database Server Internal Error**
  Internal errors that may occur if the TwinCAT Database Server is configured incorrectly.

- **TC3 Database Server Database Error**
  Database errors that may occur during communication with the corresponding databases. The different database-specific error codes are mapped in a database error list. The database-specific codes are written into the ErrorLog, as required.

- **TC3 Database Server ADS Device Error**
  ADS error that may occur during internal communication with configured ADS devices.

- **TC3 Database Server NoSQL Error**
  Database error of a NoSQL database that occurred during communication with the corresponding databases.
8.1.1.2 ADS return codes

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipResult</td>
<td>I_TcResultEvent</td>
<td>16#D6EFD6C8</td>
</tr>
<tr>
<td>EventClass</td>
<td>WSTRING(255)</td>
<td>&quot;TC3 ADS Error&quot;</td>
</tr>
<tr>
<td>EventID</td>
<td>UDINT</td>
<td>1861</td>
</tr>
<tr>
<td>EventSeverity</td>
<td>TCEVENTSEVERITY</td>
<td>TCEVENTSEVERITY_ERROR</td>
</tr>
<tr>
<td>EventMsg</td>
<td>WSTRING(255)</td>
<td>&quot;timeout elapsed&quot;</td>
</tr>
</tbody>
</table>

Global error codes
## Appendix

### Hex Dec HRESULT Name Description

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0</td>
<td>0x9811 0000</td>
<td>ERR_NOERROR</td>
<td>No error.</td>
</tr>
<tr>
<td>0x1</td>
<td>1</td>
<td>0x9811 0001</td>
<td>ERR_INTERNAL</td>
<td>Internal error.</td>
</tr>
<tr>
<td>0x2</td>
<td>2</td>
<td>0x9811 0002</td>
<td>ERR_NORTIME</td>
<td>No real-time.</td>
</tr>
<tr>
<td>0x3</td>
<td>3</td>
<td>0x9811 0003</td>
<td>ERR_ALLOCLOCKEDMEM</td>
<td>Allocation locked – memory error.</td>
</tr>
<tr>
<td>0x4</td>
<td>4</td>
<td>0x9811 0004</td>
<td>ERR_INSERTMAILBOX</td>
<td>Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.</td>
</tr>
<tr>
<td>0x5</td>
<td>5</td>
<td>0x9811 0005</td>
<td>ERR_WRONG_RECEIVE_HMSG</td>
<td>Wrong HMSG.</td>
</tr>
<tr>
<td>0x6</td>
<td>6</td>
<td>0x9811 0006</td>
<td>ERR_TARGETPORTNOTFOUND</td>
<td>Target port not found – ADS server is not started or is not reachable.</td>
</tr>
<tr>
<td>0x7</td>
<td>7</td>
<td>0x9811 0007</td>
<td>ERR_TARGETMACHINE_NOTFOUND</td>
<td>Target computer not found – AMS route was not found.</td>
</tr>
<tr>
<td>0x8</td>
<td>8</td>
<td>0x9811 0008</td>
<td>ERR_UNKNOWN_CMDID</td>
<td>Unknown command ID.</td>
</tr>
<tr>
<td>0x9</td>
<td>9</td>
<td>0x9811 0009</td>
<td>ERR_BAD_TASKID</td>
<td>Invalid task ID.</td>
</tr>
<tr>
<td>0xA</td>
<td>10</td>
<td>0x9811 000A</td>
<td>ERR_NOIO</td>
<td>No IO.</td>
</tr>
<tr>
<td>0xB</td>
<td>11</td>
<td>0x9811 000B</td>
<td>ERR_UNKNOWN_AMS_CMD</td>
<td>Unknown AMS command.</td>
</tr>
<tr>
<td>0xC</td>
<td>12</td>
<td>0x9811 000C</td>
<td>ERR_WIN32ERROR</td>
<td>Win32 error.</td>
</tr>
<tr>
<td>0xD</td>
<td>13</td>
<td>0x9811 000D</td>
<td>ERR_PORTNOTCONNECTED</td>
<td>Port not connected.</td>
</tr>
<tr>
<td>0xE</td>
<td>14</td>
<td>0x9811 000E</td>
<td>ERR_INVALID_AMS_LENGTH</td>
<td>Invalid AMS length.</td>
</tr>
<tr>
<td>0xF</td>
<td>15</td>
<td>0x9811 000F</td>
<td>ERR_INVALID_AMS_NET_ID</td>
<td>Invalid AMS Net ID.</td>
</tr>
<tr>
<td>0x10</td>
<td>16</td>
<td>0x9811 0010</td>
<td>ERR_LOWINSTLEVEL</td>
<td>Installation level is too low – TwinCAT 2 license error.</td>
</tr>
<tr>
<td>0x11</td>
<td>17</td>
<td>0x9811 0011</td>
<td>ERR_NODEBUG_INT_AVAILABLE</td>
<td>No debugging available.</td>
</tr>
<tr>
<td>0x12</td>
<td>18</td>
<td>0x9811 0012</td>
<td>ERR_PORT_DISABLED</td>
<td>Port disabled – TwinCAT system service not started.</td>
</tr>
<tr>
<td>0x13</td>
<td>19</td>
<td>0x9811 0013</td>
<td>ERR_PORT_ALREADY_CONNECTED</td>
<td>Port already connected.</td>
</tr>
<tr>
<td>0x14</td>
<td>20</td>
<td>0x9811 0014</td>
<td>ERR_AMS_SYNC_W32ERROR</td>
<td>AMS Sync Win32 error.</td>
</tr>
<tr>
<td>0x15</td>
<td>21</td>
<td>0x9811 0015</td>
<td>ERR_AMS_SYNC_TIMEOUT</td>
<td>AMS Sync Timeout.</td>
</tr>
<tr>
<td>0x16</td>
<td>22</td>
<td>0x9811 0016</td>
<td>ERR_AMS_SYNC_AMSERROR</td>
<td>AMS Sync error.</td>
</tr>
<tr>
<td>0x17</td>
<td>23</td>
<td>0x9811 0017</td>
<td>ERR_AMS_SYNC_NO_INDEX_MAP</td>
<td>No index map for AMS Sync available.</td>
</tr>
<tr>
<td>0x18</td>
<td>24</td>
<td>0x9811 0018</td>
<td>ERR_INVALID_AMS_PORT</td>
<td>Invalid AMS port.</td>
</tr>
<tr>
<td>0x19</td>
<td>25</td>
<td>0x9811 0019</td>
<td>ERR_NO_MEMORY</td>
<td>No memory.</td>
</tr>
<tr>
<td>0x1A</td>
<td>26</td>
<td>0x9811 001A</td>
<td>ERR_TCP_SEND</td>
<td>TCP send error.</td>
</tr>
<tr>
<td>0x1B</td>
<td>27</td>
<td>0x9811 001B</td>
<td>ERR_HOST_UNREACHABLE</td>
<td>Host unreachable.</td>
</tr>
<tr>
<td>0x1C</td>
<td>28</td>
<td>0x9811 001C</td>
<td>ERR_INVALID_AMS_FRAGMENT</td>
<td>Invalid AMS fragment.</td>
</tr>
<tr>
<td>0x1D</td>
<td>29</td>
<td>0x9811 001D</td>
<td>ERR_TLS_SEND</td>
<td>TLS send error – secure ADS connection failed.</td>
</tr>
<tr>
<td>0x1E</td>
<td>30</td>
<td>0x9811 001E</td>
<td>ERR_ACCESS_DENIED</td>
<td>Access denied – secure ADS access denied.</td>
</tr>
</tbody>
</table>

### RTime error codes
### Router error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x500</td>
<td>1280</td>
<td>0x9811 0500</td>
<td>ROUTERERR_NOLOCKEDMEMORY</td>
<td>Locked memory cannot be allocated.</td>
</tr>
<tr>
<td>0x501</td>
<td>1281</td>
<td>0x9811 0501</td>
<td>ROUTERERR_RESIZEMEMORY</td>
<td>The router memory size could not be changed.</td>
</tr>
<tr>
<td>0x502</td>
<td>1282</td>
<td>0x9811 0502</td>
<td>ROUTERERR_MAILBOXFULL</td>
<td>The mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x503</td>
<td>1283</td>
<td>0x9811 0503</td>
<td>ROUTERERR_DEBUGBOXFULL</td>
<td>The Debug mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x504</td>
<td>1284</td>
<td>0x9811 0504</td>
<td>ROUTERERR_UNKNOWNPORTTYPE</td>
<td>The port type is unknown.</td>
</tr>
<tr>
<td>0x505</td>
<td>1285</td>
<td>0x9811 0505</td>
<td>ROUTERERR_NOTINITIALIZED</td>
<td>The router is not initialized.</td>
</tr>
<tr>
<td>0x506</td>
<td>1286</td>
<td>0x9811 0506</td>
<td>ROUTERERR_PORTALREADYINUSE</td>
<td>The port number is already assigned.</td>
</tr>
<tr>
<td>0x507</td>
<td>1287</td>
<td>0x9811 0507</td>
<td>ROUTERERR_NOTREGISTERED</td>
<td>The port is not registered.</td>
</tr>
<tr>
<td>0x508</td>
<td>1288</td>
<td>0x9811 0508</td>
<td>ROUTERERR_NOMOREQUEUES</td>
<td>The maximum number of ports has been reached.</td>
</tr>
<tr>
<td>0x509</td>
<td>1289</td>
<td>0x9811 0509</td>
<td>ROUTERERR_INVALIDPORT</td>
<td>The port is invalid.</td>
</tr>
<tr>
<td>0x50A</td>
<td>1290</td>
<td>0x9811 050A</td>
<td>ROUTERERR_NOTACTIVATED</td>
<td>The router is not active.</td>
</tr>
<tr>
<td>0x50B</td>
<td>1291</td>
<td>0x9811 050B</td>
<td>ROUTERERR_FRAGMENTBOXFULL</td>
<td>The mailbox has reached the maximum number for fragmented messages.</td>
</tr>
<tr>
<td>0x50C</td>
<td>1292</td>
<td>0x9811 050C</td>
<td>ROUTERERR_FRAGMENTTIMEOUT</td>
<td>A fragment timeout has occurred.</td>
</tr>
<tr>
<td>0x50D</td>
<td>1293</td>
<td>0x9811 050D</td>
<td>ROUTERERR_TOBEREMOVED</td>
<td>The port is removed.</td>
</tr>
</tbody>
</table>

### General ADS error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>4096</td>
<td>0x9811 1000</td>
<td>RTERR_INTERNAL</td>
<td>Internal error in the real-time system.</td>
</tr>
<tr>
<td>0x1001</td>
<td>4097</td>
<td>0x9811 1001</td>
<td>RTERR_BADTIMPERIODS</td>
<td>Timer value is not valid.</td>
</tr>
<tr>
<td>0x1002</td>
<td>4098</td>
<td>0x9811 1002</td>
<td>RTERR_INVALIDTASKPTR</td>
<td>Task pointer has the invalid value 0 (zero).</td>
</tr>
<tr>
<td>0x1003</td>
<td>4099</td>
<td>0x9811 1003</td>
<td>RTERR_INVALIDSTACKPTR</td>
<td>Stack pointer has the invalid value 0 (zero).</td>
</tr>
<tr>
<td>0x1004</td>
<td>4100</td>
<td>0x9811 1004</td>
<td>RTERR_PRIORIGHTS</td>
<td>The request task priority is already assigned.</td>
</tr>
<tr>
<td>0x1005</td>
<td>4101</td>
<td>0x9811 1005</td>
<td>RTERR_NOMORETCB</td>
<td>No free TCB (Task Control Block) available. The maximum number of TCBs is 64.</td>
</tr>
<tr>
<td>0x1006</td>
<td>4102</td>
<td>0x9811 1006</td>
<td>RTERR_NOMORESEMAS</td>
<td>No free semaphores available. The maximum number of semaphores is 64.</td>
</tr>
<tr>
<td>0x1007</td>
<td>4103</td>
<td>0x9811 1007</td>
<td>RTERR_NOMOREQUEUES</td>
<td>No free space available in the queue. The maximum number of positions in the queue is 64.</td>
</tr>
<tr>
<td>0x100D</td>
<td>4109</td>
<td>0x9811 100D</td>
<td>RTERR_EXTIRQALREADYDEF</td>
<td>An external synchronization interrupt is already applied.</td>
</tr>
<tr>
<td>0x100E</td>
<td>4110</td>
<td>0x9811 100E</td>
<td>RTERR_EXTIRQNOTDEF</td>
<td>No external sync interrupt applied.</td>
</tr>
<tr>
<td>0x100F</td>
<td>4111</td>
<td>0x9811 100F</td>
<td>RTERR_EXTIRQINSTALLFAILED</td>
<td>Application of the external interrupt has failed.</td>
</tr>
<tr>
<td>0x1010</td>
<td>4112</td>
<td>0x9811 1010</td>
<td>RTERR_IRQNLLESSOREQUAL</td>
<td>Call of a service function in the wrong context.</td>
</tr>
<tr>
<td>0x1017</td>
<td>4119</td>
<td>0x9811 1017</td>
<td>RTERR_VMXNOTSUPPORTED</td>
<td>Intel VT-x extension is not supported.</td>
</tr>
<tr>
<td>0x1018</td>
<td>4120</td>
<td>0x9811 1018</td>
<td>RTERR_VMXDISABLED</td>
<td>Intel VT-x extension is not enabled in the BIOS.</td>
</tr>
<tr>
<td>0x1019</td>
<td>4121</td>
<td>0x9811 1019</td>
<td>RTERR_VMXCONTROLSMISSING</td>
<td>Missing function in Intel VT-x extension.</td>
</tr>
<tr>
<td>0x101A</td>
<td>4122</td>
<td>0x9811 101A</td>
<td>RTERR_VMXENABLEFAILS</td>
<td>Activation of Intel VT-x fails.</td>
</tr>
</tbody>
</table>

### General ADS error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x500</td>
<td>1280</td>
<td>0x9811 0500</td>
<td>ROUTERERR_NOLOCKEDMEMORY</td>
<td>Locked memory cannot be allocated.</td>
</tr>
<tr>
<td>0x501</td>
<td>1281</td>
<td>0x9811 0501</td>
<td>ROUTERERR_RESIZEMEMORY</td>
<td>The router memory size could not be changed.</td>
</tr>
<tr>
<td>0x502</td>
<td>1282</td>
<td>0x9811 0502</td>
<td>ROUTERERR_MAILBOXFULL</td>
<td>The mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x503</td>
<td>1283</td>
<td>0x9811 0503</td>
<td>ROUTERERR_DEBUGBOXFULL</td>
<td>The Debug mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x504</td>
<td>1284</td>
<td>0x9811 0504</td>
<td>ROUTERERR_UNKNOWNPORTTYPE</td>
<td>The port type is unknown.</td>
</tr>
<tr>
<td>0x505</td>
<td>1285</td>
<td>0x9811 0505</td>
<td>ROUTERERR_NOTINITIALIZED</td>
<td>The router is not initialized.</td>
</tr>
<tr>
<td>0x506</td>
<td>1286</td>
<td>0x9811 0506</td>
<td>ROUTERERR_PORTALREADYINUSE</td>
<td>The port number is already assigned.</td>
</tr>
<tr>
<td>0x507</td>
<td>1287</td>
<td>0x9811 0507</td>
<td>ROUTERERR_NOTREGISTERED</td>
<td>The port is not registered.</td>
</tr>
<tr>
<td>0x508</td>
<td>1288</td>
<td>0x9811 0508</td>
<td>ROUTERERR_NOMOREQUEUES</td>
<td>The maximum number of ports has been reached.</td>
</tr>
<tr>
<td>0x509</td>
<td>1289</td>
<td>0x9811 0509</td>
<td>ROUTERERR_INVALIDPORT</td>
<td>The port is invalid.</td>
</tr>
<tr>
<td>0x50A</td>
<td>1290</td>
<td>0x9811 050A</td>
<td>ROUTERERR_NOTACTIVATED</td>
<td>The router is not active.</td>
</tr>
<tr>
<td>0x50B</td>
<td>1291</td>
<td>0x9811 050B</td>
<td>ROUTERERR_FRAGMENTBOXFULL</td>
<td>The mailbox has reached the maximum number for fragmented messages.</td>
</tr>
<tr>
<td>0x50C</td>
<td>1292</td>
<td>0x9811 050C</td>
<td>ROUTERERR_FRAGMENTTIMEOUT</td>
<td>A fragment timeout has occurred.</td>
</tr>
<tr>
<td>0x50D</td>
<td>1293</td>
<td>0x9811 050D</td>
<td>ROUTERERR_TOBEREMOVED</td>
<td>The port is removed.</td>
</tr>
<tr>
<td>Hex</td>
<td>Dec</td>
<td>HRESULT</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------------------</td>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x700</td>
<td>1792</td>
<td>0x9811 0700</td>
<td>ADSERR_DEVICE_ERROR</td>
<td>General device error.</td>
</tr>
<tr>
<td>0x701</td>
<td>1793</td>
<td>0x9811 0701</td>
<td>ADSERR_DEVICE_SRVNOTSUPP</td>
<td>Service is not supported by the server.</td>
</tr>
<tr>
<td>0x702</td>
<td>1794</td>
<td>0x9811 0702</td>
<td>ADSERR_DEVICE_INVALIDGRP</td>
<td>Invalid index group.</td>
</tr>
<tr>
<td>0x703</td>
<td>1795</td>
<td>0x9811 0703</td>
<td>ADSERR_DEVICE_INVALIDOFFSET</td>
<td>Invalid index offset.</td>
</tr>
<tr>
<td>0x704</td>
<td>1796</td>
<td>0x9811 0704</td>
<td>ADSERR_DEVICE_INVALIDACCESS</td>
<td>Reading or writing not permitted.</td>
</tr>
<tr>
<td>0x705</td>
<td>1797</td>
<td>0x9811 0705</td>
<td>ADSERR_DEVICE_INVALIDSIZE</td>
<td>Parameter size not correct.</td>
</tr>
<tr>
<td>0x706</td>
<td>1798</td>
<td>0x9811 0706</td>
<td>ADSERRDEVICE_INVALIDDATA</td>
<td>Invalid data values.</td>
</tr>
<tr>
<td>0x707</td>
<td>1799</td>
<td>0x9811 0707</td>
<td>ADSERR_DEVICE_NotreADY</td>
<td>Device is not ready to operate.</td>
</tr>
<tr>
<td>0x708</td>
<td>1800</td>
<td>0x9811 0708</td>
<td>ADSERR_DEVICE_BUSY</td>
<td>Device is busy.</td>
</tr>
<tr>
<td>0x709</td>
<td>1801</td>
<td>0x9811 0709</td>
<td>ADSERR_DEVICE_INVALIDCONTEXT</td>
<td>Invalid operating system context. This can result from use of ADS function blocks in different tasks. It may be possible to resolve this through Multi-task data access synchronization in the PLC.</td>
</tr>
<tr>
<td>0x70A</td>
<td>1802</td>
<td>0x9811 070A</td>
<td>ADSERR_DEVICE_NOMEMORY</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td>0x70B</td>
<td>1803</td>
<td>0x9811 070B</td>
<td>ADSERR_DEVICE_INVALID Parm</td>
<td>Invalid parameter values.</td>
</tr>
<tr>
<td>0x70C</td>
<td>1804</td>
<td>0x9811 070C</td>
<td>ADSERR_DEVICE_NOTFOUND</td>
<td>Not found (files,...).</td>
</tr>
<tr>
<td>0x70D</td>
<td>1805</td>
<td>0x9811 070D</td>
<td>ADSERR_DEVICE_SYNTAX</td>
<td>Syntax error in file or command.</td>
</tr>
<tr>
<td>0x70E</td>
<td>1806</td>
<td>0x9811 070E</td>
<td>ADSERRDEVICE_INCOMPATIBLE</td>
<td>Objects do not match.</td>
</tr>
<tr>
<td>0x70F</td>
<td>1807</td>
<td>0x9811 070F</td>
<td>ADSERR_DEVICE_EXISTS</td>
<td>Object already exists.</td>
</tr>
<tr>
<td>0x710</td>
<td>1808</td>
<td>0x9811 0710</td>
<td>ADSERR_DEVICE_SYMBOLNOTFOUND</td>
<td>Symbol not found.</td>
</tr>
<tr>
<td>0x711</td>
<td>1809</td>
<td>0x9811 0711</td>
<td>ADSERR_DEVICE_SYMBOLVERSIONINVALID</td>
<td>Invalid symbol version. This can occur due to an online change. Create a new handle.</td>
</tr>
<tr>
<td>0x712</td>
<td>1810</td>
<td>0x9811 0712</td>
<td>ADSERR_DEVICE_INVALIDSTATE</td>
<td>Device (server) is invalid state.</td>
</tr>
<tr>
<td>0x713</td>
<td>1811</td>
<td>0x9811 0713</td>
<td>ADSERR_DEVICE_TRANSMODENOTSUPP</td>
<td>AdsTransMode not supported.</td>
</tr>
<tr>
<td>0x714</td>
<td>1812</td>
<td>0x9811 0714</td>
<td>ADSERR_DEVICE_NOTIFYHINDINVALID</td>
<td>Notification handle is invalid.</td>
</tr>
<tr>
<td>0x715</td>
<td>1813</td>
<td>0x9811 0715</td>
<td>ADSERR_DEVICE_CLIENTUNKNOWN</td>
<td>Notification client not registered.</td>
</tr>
<tr>
<td>0x716</td>
<td>1814</td>
<td>0x9811 0716</td>
<td>ADSERR_DEVICE_NOMOREHDLs</td>
<td>No further handle available.</td>
</tr>
<tr>
<td>0x717</td>
<td>1815</td>
<td>0x9811 0717</td>
<td>ADSERR_DEVICE_INVALIDWATCHSIZE</td>
<td>Notification size too large.</td>
</tr>
<tr>
<td>0x718</td>
<td>1816</td>
<td>0x9811 0718</td>
<td>ADSERR_DEVICE_NOTINIT</td>
<td>Device not initialized.</td>
</tr>
<tr>
<td>0x719</td>
<td>1817</td>
<td>0x9811 0719</td>
<td>ADSERR_DEVICE_TIMEOUT</td>
<td>Device has a timeout.</td>
</tr>
<tr>
<td>0x71A</td>
<td>1818</td>
<td>0x9811 071A</td>
<td>ADSERR_DEVICE_NOINTERFACE</td>
<td>Interface query failed.</td>
</tr>
<tr>
<td>0x71B</td>
<td>1819</td>
<td>0x9811 071B</td>
<td>ADSERR_DEVICE_INVALIDINTERFACE</td>
<td>Wrong interface requested.</td>
</tr>
<tr>
<td>0x71C</td>
<td>1820</td>
<td>0x9811 071C</td>
<td>ADSERR_DEVICE_INVALIDCLSID</td>
<td>Class ID is invalid.</td>
</tr>
<tr>
<td>0x71D</td>
<td>1821</td>
<td>0x9811 071D</td>
<td>ADSERR_DEVICE_INVALIDOBJID</td>
<td>Object ID is invalid.</td>
</tr>
<tr>
<td>0x71E</td>
<td>1822</td>
<td>0x9811 071E</td>
<td>ADSERR_DEVICE_PENDING</td>
<td>Request pending.</td>
</tr>
<tr>
<td>0x71F</td>
<td>1823</td>
<td>0x9811 071F</td>
<td>ADSERR_DEVICE_ABORTED</td>
<td>Request is aborted.</td>
</tr>
<tr>
<td>0x720</td>
<td>1824</td>
<td>0x9811 0720</td>
<td>ADSERR_DEVICE_WARNING</td>
<td>Signal warning.</td>
</tr>
<tr>
<td>0x721</td>
<td>1825</td>
<td>0x9811 0721</td>
<td>ADSERR_DEVICE_INVALIDDARRAYIDX</td>
<td>Invalid array index.</td>
</tr>
<tr>
<td>0x722</td>
<td>1826</td>
<td>0x9811 0722</td>
<td>ADSERRDEVICE_SYMBOLNOTACTIVE</td>
<td>Symbol not active.</td>
</tr>
<tr>
<td>0x723</td>
<td>1827</td>
<td>0x9811 0723</td>
<td>ADSERR_DEVICE_ACCESSDENIED</td>
<td>Access denied.</td>
</tr>
<tr>
<td>0x724</td>
<td>1828</td>
<td>0x9811 0724</td>
<td>ADSERR_DEVICE_LICENSENOTFOUND</td>
<td>Missing license.</td>
</tr>
<tr>
<td>0x725</td>
<td>1829</td>
<td>0x9811 0725</td>
<td>ADSERR_DEVICE_LICENSEEXPIRED</td>
<td>License expired.</td>
</tr>
<tr>
<td>0x726</td>
<td>1830</td>
<td>0x9811 0726</td>
<td>ADSERR_DEVICE_LICENSEEXCEEDED</td>
<td>License exceeded.</td>
</tr>
<tr>
<td>0x727</td>
<td>1831</td>
<td>0x9811 0727</td>
<td>ADSERR_DEVICE_LICENSEINVALID</td>
<td>Invalid license.</td>
</tr>
<tr>
<td>0x728</td>
<td>1832</td>
<td>0x9811 0728</td>
<td>ADSERR_DEVICE_LICENSESYSTEMID</td>
<td>License problem: System ID is invalid.</td>
</tr>
<tr>
<td>0x729</td>
<td>1833</td>
<td>0x9811 0729</td>
<td>ADSERR_DEVICE_LICENSEENOTLIMIT</td>
<td>License not limited in time.</td>
</tr>
<tr>
<td>0x72A</td>
<td>1834</td>
<td>0x9811 072A</td>
<td>ADSERR_DEVICE_LICENSEFUTUREISSUE</td>
<td>License problem: Time in the future.</td>
</tr>
<tr>
<td>0x72B</td>
<td>1835</td>
<td>0x9811 072B</td>
<td>ADSERR_DEVICE_LICENSEMETOLONG</td>
<td>License period too long.</td>
</tr>
<tr>
<td>0x72C</td>
<td>1836</td>
<td>0x9811 072C</td>
<td>ADSERR_DEVICE_EXCEPTION</td>
<td>Exception at system startup.</td>
</tr>
<tr>
<td>0x72D</td>
<td>1837</td>
<td>0x9811 072D</td>
<td>ADSERR_DEVICE_LICENSEDEPULICATED</td>
<td>License file read twice.</td>
</tr>
<tr>
<td>0x72E</td>
<td>1838</td>
<td>0x9811 072E</td>
<td>ADSERR_DEVICE_LICENSESIGNATUREINVALID</td>
<td>Invalid signature.</td>
</tr>
<tr>
<td>0x72F</td>
<td>1839</td>
<td>0x9811 072F</td>
<td>ADSERR_DEVICE_CERTIFICATEINVALID</td>
<td>Invalid certificate.</td>
</tr>
<tr>
<td>0x730</td>
<td>1840</td>
<td>0x9811 0730</td>
<td>ADSERR_DEVICE_LICENSEOEMNOTFOUND</td>
<td>Public key not known from OEM.</td>
</tr>
<tr>
<td>0x731</td>
<td>1841</td>
<td>0x9811 0731</td>
<td>ADSERR_DEVICE_LICENSERESTRICTED</td>
<td>License not valid for this system ID.</td>
</tr>
<tr>
<td>0x732</td>
<td>1842</td>
<td>0x9811 0732</td>
<td>ADSERR_DEVICE_LICENSEDEMODENIED</td>
<td>Demo license prohibited.</td>
</tr>
<tr>
<td>0x733</td>
<td>1843</td>
<td>0x9811 0733</td>
<td>ADSERR_DEVICE_INVALIDFCID</td>
<td>Invalid function ID.</td>
</tr>
<tr>
<td>0x734</td>
<td>1844</td>
<td>0x9811 0734</td>
<td>ADSERR_DEVICE_OUTOF RANGE</td>
<td>Outside the valid range.</td>
</tr>
<tr>
<td>0x735</td>
<td>1845</td>
<td>0x9811 0735</td>
<td>ADSERR_DEVICE_INVALIDALIGNMENT</td>
<td>Invalid alignment.</td>
</tr>
</tbody>
</table>
### Database return codes

#### MAIN

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipResult</td>
<td>l_TcResultEvent</td>
<td>16#D03F0</td>
</tr>
<tr>
<td>EventClass</td>
<td>WSTRING(255)</td>
<td>&quot;TC3 Database Server Database Error&quot;</td>
</tr>
<tr>
<td>EventID</td>
<td>UINT</td>
<td>61</td>
</tr>
<tr>
<td>EventSeverity</td>
<td>TCEVENTSEVERITY</td>
<td>TCEVENTSEVERITY_ERROR</td>
</tr>
<tr>
<td>EventMsg</td>
<td>WSTRING(255)</td>
<td>&quot;SQLSkate 42502 Base table or view not found&quot;</td>
</tr>
</tbody>
</table>

```c
17. ipResult[16#D03F0] := fbdLDBWrite.ip7CResultEvent[16#D03F0];
18. EventClass["Tc3Database"] := ipResult.EventClassDisplayName;
```
<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>ErrorName</th>
<th>ErrorDescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB_SQLState_01000</td>
<td>General warning</td>
</tr>
<tr>
<td>2</td>
<td>DB_SQLState_01001</td>
<td>Cursor operation conflict</td>
</tr>
<tr>
<td>3</td>
<td>DB_SQLState_01002</td>
<td>Disconnect error</td>
</tr>
<tr>
<td>4</td>
<td>DB_SQLState_01003</td>
<td>NULL value eliminated in set function</td>
</tr>
<tr>
<td>5</td>
<td>DB_SQLState_01004</td>
<td>String data, right-truncated</td>
</tr>
<tr>
<td>6</td>
<td>DB_SQLState_01006</td>
<td>Privilege not revoked</td>
</tr>
<tr>
<td>7</td>
<td>DB_SQLState_01007</td>
<td>Privilege not granted</td>
</tr>
<tr>
<td>8</td>
<td>DB_SQLState_01S00</td>
<td>Invalid connection string attribute</td>
</tr>
<tr>
<td>9</td>
<td>DB_SQLState_01S01</td>
<td>Error in row</td>
</tr>
<tr>
<td>10</td>
<td>DB_SQLState_01S02</td>
<td>Option value changed</td>
</tr>
<tr>
<td>11</td>
<td>DB_SQLState_01S06</td>
<td>Attempt to fetch before the result set returned the first row set</td>
</tr>
<tr>
<td>12</td>
<td>DB_SQLState_01S07</td>
<td>Fractional truncation</td>
</tr>
<tr>
<td>13</td>
<td>DB_SQLState_01S08</td>
<td>Error saving File DSN</td>
</tr>
<tr>
<td>14</td>
<td>DB_SQLState_01S09</td>
<td>Invalid keyword</td>
</tr>
<tr>
<td>15</td>
<td>DB_SQLState_07000</td>
<td>Dynamic SQL error</td>
</tr>
<tr>
<td>16</td>
<td>DB_SQLState_07001</td>
<td>Wrong number of parameters</td>
</tr>
<tr>
<td>17</td>
<td>DB_SQLState_07002</td>
<td>COUNT field incorrect</td>
</tr>
<tr>
<td>18</td>
<td>DB_SQLState_07005</td>
<td>Prepared statement not a cursor-specification</td>
</tr>
<tr>
<td>19</td>
<td>DB_SQLState_07006</td>
<td>Restricted data type attribute violation</td>
</tr>
<tr>
<td>20</td>
<td>DB_SQLState_07009</td>
<td>Invalid descriptor index</td>
</tr>
<tr>
<td>21</td>
<td>DB_SQLState_07S01</td>
<td>Invalid use of default parameter</td>
</tr>
<tr>
<td>22</td>
<td>DB_SQLState_08000</td>
<td>Connection exception</td>
</tr>
<tr>
<td>23</td>
<td>DB_SQLState_08001</td>
<td>Client unable to establish connection</td>
</tr>
<tr>
<td>24</td>
<td>DB_SQLState_08002</td>
<td>Connection name in use</td>
</tr>
<tr>
<td>25</td>
<td>DB_SQLState_08003</td>
<td>Connection not open</td>
</tr>
<tr>
<td>26</td>
<td>DB_SQLState_08004</td>
<td>Server rejected the connection</td>
</tr>
<tr>
<td>27</td>
<td>DB_SQLState_08007</td>
<td>Connection failure during transaction</td>
</tr>
<tr>
<td>28</td>
<td>DB_SQLState_08S01</td>
<td>Communication link failure</td>
</tr>
<tr>
<td>29</td>
<td>DB_SQLState_21000</td>
<td>Cardinality violation</td>
</tr>
<tr>
<td>30</td>
<td>DB_SQLState_21S01</td>
<td>Insert value list does not match column list</td>
</tr>
<tr>
<td>31</td>
<td>DB_SQLState_21S02</td>
<td>Degree of derived table does not match column list</td>
</tr>
<tr>
<td>32</td>
<td>DB_SQLState_22000</td>
<td>Data exception</td>
</tr>
<tr>
<td>33</td>
<td>DB_SQLState_22001</td>
<td>String data, right-truncated</td>
</tr>
<tr>
<td>34</td>
<td>DB_SQLState_22002</td>
<td>Indicator variable required but not supplied</td>
</tr>
<tr>
<td>35</td>
<td>DB_SQLState_22003</td>
<td>Numeric value out of range</td>
</tr>
<tr>
<td>36</td>
<td>DB_SQLState_22007</td>
<td>Invalid datetime format</td>
</tr>
<tr>
<td>37</td>
<td>DB_SQLState_22008</td>
<td>Date/time field overflow</td>
</tr>
<tr>
<td>38</td>
<td>DB_SQLState_22012</td>
<td>Division by zero</td>
</tr>
<tr>
<td>39</td>
<td>DB_SQLState_22015</td>
<td>Interval field overflow</td>
</tr>
<tr>
<td>40</td>
<td>DB_SQLState_22018</td>
<td>Invalid character value for cast specification</td>
</tr>
<tr>
<td>41</td>
<td>DB_SQLState_22019</td>
<td>Invalid escape character</td>
</tr>
<tr>
<td>42</td>
<td>DB_SQLState_22025</td>
<td>Invalid escape sequence</td>
</tr>
<tr>
<td>43</td>
<td>DB_SQLState_22026</td>
<td>String data, length mismatch</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>DB_SQLState_23000 Integrity constraint violation</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>DB_SQLState_24000 Invalid cursor state</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>DB_SQLState_25000 Invalid transaction state</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>DB_SQLState_25S01 Transaction state</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>DB_SQLState_25S02 Transaction is still active</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>DB_SQLState_25S03 Transaction is rolled back</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>DB_SQLState_28000 Invalid authorization specification</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>DB_SQLState_34000 Invalid cursor name</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>DB_SQLState_3C000 Duplicate cursor name</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>DB_SQLState_3D000 Invalid catalog name</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>DB_SQLState_3F000 Invalid schema name</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>DB_SQLState_40000 Transaction rollback</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>DB_SQLState_40001 Serialization failure</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>DB_SQLState_40002 Integrity constraint violation</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>DB_SQLState_40003 Statement completion unknown</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>DB_SQLState_42000 Syntax error or access violation</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>DB_SQLState_42S01 Base table or view already exists</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>DB_SQLState_42S02 Base table or view not found</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>DB_SQLState_42S11 Index already exists</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>DB_SQLState_42S12 Index not found</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>DB_SQLState_42S21 Column already exists</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>DB_SQLState_42S22 Column not found</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>DB_SQLState_44000 WITH CHECK OPTION violation</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>DB_SQLState_HY000 General error</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>DB_SQLState_HY001 Memory allocation error</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>DB_SQLState_HY003 Invalid application buffer type</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>DB_SQLState_HY004 Invalid SQL data type</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>DB_SQLState_HY007 Associated statement is not prepared</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>DB_SQLState_HY008 Operation cancelled</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>DB_SQLState_HY009 Invalid use of null pointer</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>DB_SQLState_HY010 Function sequence error</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>DB_SQLState_HY011 Attribute cannot be set now</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>DB_SQLState_HY012 Invalid transaction operation code</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>DB_SQLState_HY013 Memory management error</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>DB_SQLState_HY014 Limit on the number of handles exceeded</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>DB_SQLState_HY015 No cursor name available</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>DB_SQLState_HY016 Cannot modify an implementation row descriptor</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>DB_SQLState_HY017 Invalid use of an automatically allocated descriptor handle</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>DB_SQLState_HY018</td>
<td>Server declined cancel request</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>83</td>
<td>DB_SQLState_HY019</td>
<td>Non-character and non-binary data sent in pieces</td>
</tr>
<tr>
<td>84</td>
<td>DB_SQLState_HY020</td>
<td>Attempt to concatenate a null value</td>
</tr>
<tr>
<td>85</td>
<td>DB_SQLState_HY021</td>
<td>Inconsistent descriptor information</td>
</tr>
<tr>
<td>86</td>
<td>DB_SQLState_HY024</td>
<td>Invalid attribute value</td>
</tr>
<tr>
<td>87</td>
<td>DB_SQLState_HY090</td>
<td>Invalid string or buffer length</td>
</tr>
<tr>
<td>88</td>
<td>DB_SQLState_HY091</td>
<td>Invalid descriptor field identifier</td>
</tr>
<tr>
<td>89</td>
<td>DB_SQLState_HY092</td>
<td>Invalid attribute/option identifier</td>
</tr>
<tr>
<td>90</td>
<td>DB_SQLState_HY095</td>
<td>Function type out of range</td>
</tr>
<tr>
<td>91</td>
<td>DB_SQLState_HY096</td>
<td>Invalid information type</td>
</tr>
<tr>
<td>92</td>
<td>DB_SQLState_HY097</td>
<td>Column type out of range</td>
</tr>
<tr>
<td>93</td>
<td>DB_SQLState_HY098</td>
<td>Scope type out of range</td>
</tr>
<tr>
<td>94</td>
<td>DB_SQLState_HY099</td>
<td>Nullable type out of range</td>
</tr>
<tr>
<td>95</td>
<td>DB_SQLState_HY100</td>
<td>Uniqueness option type out of range</td>
</tr>
<tr>
<td>96</td>
<td>DB_SQLState_HY101</td>
<td>Accuracy option type out of range</td>
</tr>
<tr>
<td>97</td>
<td>DB_SQLState_HY103</td>
<td>Invalid retrieval code</td>
</tr>
<tr>
<td>98</td>
<td>DB_SQLState_HY104</td>
<td>Invalid precision or scale value</td>
</tr>
<tr>
<td>99</td>
<td>DB_SQLState_HY105</td>
<td>Invalid parameter type</td>
</tr>
<tr>
<td>100</td>
<td>DB_SQLState_HY106</td>
<td>Fetch type out of range</td>
</tr>
<tr>
<td>101</td>
<td>DB_SQLState_HY107</td>
<td>Row value out of range</td>
</tr>
<tr>
<td>102</td>
<td>DB_SQLState_HY109</td>
<td>Invalid cursor position</td>
</tr>
<tr>
<td>103</td>
<td>DB_SQLState_HY110</td>
<td>Invalid driver completion</td>
</tr>
<tr>
<td>104</td>
<td>DB_SQLState_HY111</td>
<td>Invalid bookmark value</td>
</tr>
<tr>
<td>105</td>
<td>DB_SQLState_HYC00</td>
<td>Optional feature not implemented</td>
</tr>
<tr>
<td>106</td>
<td>DB_SQLState_HYT00</td>
<td>Timeout expired</td>
</tr>
<tr>
<td>107</td>
<td>DB_SQLState_HYT01</td>
<td>Connection timeout expired</td>
</tr>
<tr>
<td>108</td>
<td>DB_SQLState_IM001</td>
<td>Driver does not support this function</td>
</tr>
<tr>
<td>109</td>
<td>DB_SQLState_IM002</td>
<td>Data source name not found and no default driver specified</td>
</tr>
<tr>
<td>110</td>
<td>DB_SQLState_IM003</td>
<td>Specified driver could not be loaded</td>
</tr>
<tr>
<td>111</td>
<td>DB_SQLState_IM004</td>
<td>Driver's SQLAllocHandle on SQL_HANDLE_ENV failed</td>
</tr>
<tr>
<td>112</td>
<td>DB_SQLState_IM005</td>
<td>Driver's SQLAllocHandle on SQL_HANDLE_DBC failed</td>
</tr>
<tr>
<td>113</td>
<td>DB_SQLState_IM006</td>
<td>Driver's SQLSetConnectAttr failed</td>
</tr>
<tr>
<td>114</td>
<td>DB_SQLState_IM007</td>
<td>No data source or driver specified dialog prohibited</td>
</tr>
<tr>
<td>115</td>
<td>DB_SQLState_IM008</td>
<td>Dialog failed</td>
</tr>
<tr>
<td>116</td>
<td>DB_SQLState_IM009</td>
<td>Unable to load translation DLL</td>
</tr>
<tr>
<td>117</td>
<td>DB_SQLState_IM010</td>
<td>Data source name too long</td>
</tr>
<tr>
<td>118</td>
<td>DB_SQLState_IM011</td>
<td>Driver name too long</td>
</tr>
<tr>
<td>119</td>
<td>DB_SQLState_IM012</td>
<td>DRIVER keyword syntax error</td>
</tr>
<tr>
<td>120</td>
<td>DB_SQLState_IM013</td>
<td>Trace file error</td>
</tr>
<tr>
<td>121</td>
<td>DB_SQLState_IM014</td>
<td>Invalid name of File DSN</td>
</tr>
<tr>
<td>122</td>
<td>DB_SQLState_IM015</td>
<td>Corrupt file data source</td>
</tr>
<tr>
<td>200</td>
<td>DB_FUNCTION_NOT_IMPLEMENTED</td>
<td>Function not implemented</td>
</tr>
<tr>
<td>255</td>
<td>DB_SQLState_UNSPECIFIED</td>
<td>Unspecified Error</td>
</tr>
</tbody>
</table>
8.1.1.4 NoSQL database return codes

<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>ErrorName</th>
<th>ErrorDescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NoSql_0</td>
<td>No Error</td>
</tr>
<tr>
<td>1</td>
<td>NoSql_Specific_1</td>
<td>Internal MongoDB exception. Please check the information log for more details.</td>
</tr>
<tr>
<td>2</td>
<td>NoSql_Specific_2</td>
<td>Database side error. Please check the information log for more details.</td>
</tr>
<tr>
<td>8</td>
<td>NoSql_Specific_8</td>
<td>Timeout error occurred on command execution</td>
</tr>
<tr>
<td>9</td>
<td>NoSql_Specific_9</td>
<td>Format error occurred on command execution. Please check the command or document syntax.</td>
</tr>
<tr>
<td>11</td>
<td>NoSql_Specific_11</td>
<td>RunCommand execution failed. Please check the event log for more information</td>
</tr>
<tr>
<td>12</td>
<td>NoSql_Specific_12</td>
<td>There is no metadata including the tableschema for this table</td>
</tr>
<tr>
<td>13</td>
<td>NoSql_Specific_13</td>
<td>Syntax error in command / filter</td>
</tr>
<tr>
<td>14</td>
<td>NoSql_Specific_14</td>
<td>Connection error occurred</td>
</tr>
<tr>
<td>15</td>
<td>NoSql_Specific_15</td>
<td>Authentication failed</td>
</tr>
<tr>
<td>16</td>
<td>NoSql_Specific_16</td>
<td>Error occurred during write operation</td>
</tr>
<tr>
<td>20</td>
<td>NoSql_Specific_20</td>
<td>Functionality is not supported by the TwinCAT Database Server</td>
</tr>
<tr>
<td>48</td>
<td>NoSql_Specific_48</td>
<td>Namespace / CollectionName already exists</td>
</tr>
<tr>
<td>51</td>
<td>NoSql_Specific_51</td>
<td>The queried data is empty</td>
</tr>
<tr>
<td>141</td>
<td>NoSql_Specific_141</td>
<td>Internal TwinCAT 3 Database Server Error</td>
</tr>
</tbody>
</table>

8.1.1.5 TwinCAT Database Server Codes

```c
    ipResult := fbLcdbWrite.ipToResultEvent(16#32D53F0);
    EventClass := ipResult.EventClassDisplayName;
    EventID := ipResult.EventId;
    EventSeverity := ipResult.Severity;
    EventMsg := "No Error" := ipResult.Text;
```
<table>
<thead>
<tr>
<th>ErrorCode</th>
<th>ErrorName</th>
<th>ErrorDescription</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>InternalErrorCode_0</td>
<td>No Error</td>
</tr>
<tr>
<td>1</td>
<td>InternalErrorCode_1</td>
<td>NULL Values not allowed</td>
</tr>
<tr>
<td>2</td>
<td>InternalErrorCode_2</td>
<td>FB_DBRead selected value is NULL</td>
</tr>
<tr>
<td>3</td>
<td>InternalErrorCode_3</td>
<td>DBID is unknown</td>
</tr>
<tr>
<td>4</td>
<td>InternalErrorCode_4</td>
<td>ADSDevID is unknown</td>
</tr>
<tr>
<td>5</td>
<td>InternalErrorCode_5</td>
<td>No open database connection found for DBID: xy</td>
</tr>
<tr>
<td>6</td>
<td>InternalErrorCode_6</td>
<td>No open ADS Device connection found for ADSDevID: xy</td>
</tr>
<tr>
<td>7</td>
<td>InternalErrorCode_7</td>
<td>Init of AutoLog groups failed</td>
</tr>
<tr>
<td>8</td>
<td>InternalErrorCode_8</td>
<td>AutoLog could NOT be started. TwinCAT Database Server has no valid device state</td>
</tr>
<tr>
<td>9</td>
<td>InternalErrorCode_9</td>
<td>Record select value return =&gt; Wrong Parametersize from ADS-Device</td>
</tr>
<tr>
<td>10</td>
<td>InternalErrorCode_10</td>
<td>Invalid Parameter</td>
</tr>
<tr>
<td>11</td>
<td>InternalErrorCode_11</td>
<td>Couldn’t find PLCDBWrite Value in list</td>
</tr>
<tr>
<td>12</td>
<td>InternalErrorCode_12</td>
<td>No valid symbol TYPE</td>
</tr>
<tr>
<td>13</td>
<td>InternalErrorCode_13</td>
<td>Invalid parameter size</td>
</tr>
<tr>
<td>14</td>
<td>InternalErrorCode_14</td>
<td>Execution timeout</td>
</tr>
<tr>
<td>15</td>
<td>InternalErrorCode_15</td>
<td>Database connection already open</td>
</tr>
<tr>
<td>16</td>
<td>InternalErrorCode_16</td>
<td>Database connection not initialized</td>
</tr>
<tr>
<td>1000</td>
<td>InternalErrorCode_1000</td>
<td>Unknown internal Error</td>
</tr>
</tbody>
</table>

### 8.1.2 Tc2_Database

#### 8.1.2.1 ADS Return Codes

Grouping of error codes: \(0x000\)\[394\], \(0x500\)\[395\], \(0x700\)\[396\], \(0x1000\)\[398\], ...

Global error codes
## Hex Dec HRESULT Name Description

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0</td>
<td>0x9811 0000</td>
<td>ERR_NOERROR</td>
<td>No error.</td>
</tr>
<tr>
<td>0x1</td>
<td>1</td>
<td>0x9811 0001</td>
<td>ERR_INTERNAL</td>
<td>Internal error.</td>
</tr>
<tr>
<td>0x2</td>
<td>2</td>
<td>0x9811 0002</td>
<td>ERR_NORTIME</td>
<td>No real-time.</td>
</tr>
<tr>
<td>0x3</td>
<td>3</td>
<td>0x9811 0003</td>
<td>ERR_ALLOCCLOCKEDMEM</td>
<td>Allocation locked – memory error.</td>
</tr>
<tr>
<td>0x4</td>
<td>4</td>
<td>0x9811 0004</td>
<td>ERR_INSERTMAILBOX</td>
<td>Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.</td>
</tr>
<tr>
<td>0x5</td>
<td>5</td>
<td>0x9811 0005</td>
<td>ERR_WRONGRECEIVEHMSG</td>
<td>Wrong HMSG.</td>
</tr>
<tr>
<td>0x6</td>
<td>6</td>
<td>0x9811 0006</td>
<td>ERR_TARGETPORTNOTFOUND</td>
<td>Target port not found – ADS server is not started or is not reachable.</td>
</tr>
<tr>
<td>0x7</td>
<td>7</td>
<td>0x9811 0007</td>
<td>ERR_TARGETMACHINENOTFOUND</td>
<td>Target computer not found – AMS route was not found.</td>
</tr>
<tr>
<td>0x8</td>
<td>8</td>
<td>0x9811 0008</td>
<td>ERR_UNKNOWNCMDID</td>
<td>Unknown command ID.</td>
</tr>
<tr>
<td>0x9</td>
<td>9</td>
<td>0x9811 0009</td>
<td>ERR_BADTASKID</td>
<td>Invalid task ID.</td>
</tr>
<tr>
<td>0xA</td>
<td>10</td>
<td>0x9811 000A</td>
<td>ERR_NOIO</td>
<td>No IO.</td>
</tr>
<tr>
<td>0xB</td>
<td>11</td>
<td>0x9811 000B</td>
<td>ERR_UNKNOWNAMS_CMD</td>
<td>Unknown AMS command.</td>
</tr>
<tr>
<td>0xC</td>
<td>12</td>
<td>0x9811 000C</td>
<td>ERR_WIN32ERROR</td>
<td>Win32 error.</td>
</tr>
<tr>
<td>0xD</td>
<td>13</td>
<td>0x9811 000D</td>
<td>ERR_PORTNOTCONNECTED</td>
<td>Port not connected.</td>
</tr>
<tr>
<td>0xE</td>
<td>14</td>
<td>0x9811 000E</td>
<td>ERR_INVALIDAMS_LENGTH</td>
<td>Invalid AMS length.</td>
</tr>
<tr>
<td>0xF</td>
<td>15</td>
<td>0x9811 000F</td>
<td>ERR_INVALIDAMS_NETID</td>
<td>Invalid AMS Net ID.</td>
</tr>
<tr>
<td>0x10</td>
<td>16</td>
<td>0x9811 0010</td>
<td>ERR_LOWINSTLEVEL</td>
<td>Installation level is too low—TwinCAT 2 license error.</td>
</tr>
<tr>
<td>0x11</td>
<td>17</td>
<td>0x9811 0011</td>
<td>ERR_NODEBUGINTAVAILABLE</td>
<td>No debugging available.</td>
</tr>
<tr>
<td>0x12</td>
<td>18</td>
<td>0x9811 0012</td>
<td>ERR_PORTDISABLED</td>
<td>Port disabled – TwinCAT system service not started.</td>
</tr>
<tr>
<td>0x13</td>
<td>19</td>
<td>0x9811 0013</td>
<td>ERR_PORTALREADYCONNECTED</td>
<td>Port already connected.</td>
</tr>
<tr>
<td>0x14</td>
<td>20</td>
<td>0x9811 0014</td>
<td>ERR_AMSSYNC_W32ERROR</td>
<td>AMS Sync Win32 error.</td>
</tr>
<tr>
<td>0x15</td>
<td>21</td>
<td>0x9811 0015</td>
<td>ERR_AMSSYNC_TIMEOUT</td>
<td>AMS Sync Timeout.</td>
</tr>
<tr>
<td>0x16</td>
<td>22</td>
<td>0x9811 0016</td>
<td>ERR_AMSSYNC_AMSERROR</td>
<td>AMS Sync error.</td>
</tr>
<tr>
<td>0x17</td>
<td>23</td>
<td>0x9811 0017</td>
<td>ERR_AMSSYNC_NOINDEXINMAP</td>
<td>No index map for AMS Sync available.</td>
</tr>
<tr>
<td>0x18</td>
<td>24</td>
<td>0x9811 0018</td>
<td>ERR_INVALIDAMSPORT</td>
<td>Invalid AMS port.</td>
</tr>
<tr>
<td>0x19</td>
<td>25</td>
<td>0x9811 0019</td>
<td>ERR_NOMEMORY</td>
<td>No memory.</td>
</tr>
<tr>
<td>0x1A</td>
<td>26</td>
<td>0x9811 001A</td>
<td>ERR_TCPSEND</td>
<td>TCP send error.</td>
</tr>
<tr>
<td>0x1B</td>
<td>27</td>
<td>0x9811 001B</td>
<td>ERR_HOSTUNREACHABLE</td>
<td>Host unreachable.</td>
</tr>
<tr>
<td>0x1C</td>
<td>28</td>
<td>0x9811 001C</td>
<td>ERR_INVALIDAMSFRAAGMENT</td>
<td>Invalid AMS fragment.</td>
</tr>
<tr>
<td>0x1D</td>
<td>29</td>
<td>0x9811 001D</td>
<td>ERR_TLSSEND</td>
<td>TLS send error – secure ADS connection failed.</td>
</tr>
<tr>
<td>0x1E</td>
<td>30</td>
<td>0x9811 001E</td>
<td>ERR_ACCESSDENIED</td>
<td>Access denied – secure ADS access denied.</td>
</tr>
</tbody>
</table>

### Router error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>HRESULT</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x500</td>
<td>1280</td>
<td>0x9811 0500</td>
<td>ROUTERERR_NOLOCKEDMEMORY</td>
<td>Locked memory cannot be allocated.</td>
</tr>
<tr>
<td>0x501</td>
<td>1281</td>
<td>0x9811 0501</td>
<td>ROUTERERR_RESIZEMEMORY</td>
<td>The router memory size could not be changed.</td>
</tr>
<tr>
<td>0x502</td>
<td>1282</td>
<td>0x9811 0502</td>
<td>ROUTERERR_MAILBOXFULL</td>
<td>The mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x503</td>
<td>1283</td>
<td>0x9811 0503</td>
<td>ROUTERERR_DEBUGBOXFULL</td>
<td>The Debug mailbox has reached the maximum number of possible messages.</td>
</tr>
<tr>
<td>0x504</td>
<td>1284</td>
<td>0x9811 0504</td>
<td>ROUTERERR_UNKNOWNPORTTYPE</td>
<td>The port type is unknown.</td>
</tr>
<tr>
<td>0x505</td>
<td>1285</td>
<td>0x9811 0505</td>
<td>ROUTERERR_NOTINITIALIZED</td>
<td>The router is not initialized.</td>
</tr>
<tr>
<td>0x506</td>
<td>1286</td>
<td>0x9811 0506</td>
<td>ROUTERERR_PORTALREADYINUSE</td>
<td>The port number is already assigned.</td>
</tr>
<tr>
<td>0x507</td>
<td>1287</td>
<td>0x9811 0507</td>
<td>ROUTERERR_NOTREGISTERED</td>
<td>The port is not registered.</td>
</tr>
<tr>
<td>0x508</td>
<td>1288</td>
<td>0x9811 0508</td>
<td>ROUTERERR_NOMOREQUEUES</td>
<td>The maximum number of ports has been reached.</td>
</tr>
<tr>
<td>0x509</td>
<td>1289</td>
<td>0x9811 0509</td>
<td>ROUTERERR_INVALIDPORT</td>
<td>The port is invalid.</td>
</tr>
<tr>
<td>0x50A</td>
<td>1290</td>
<td>0x9811 050A</td>
<td>ROUTERERR_NOTACTIVATED</td>
<td>The router is not active.</td>
</tr>
<tr>
<td>0x50B</td>
<td>1291</td>
<td>0x9811 050B</td>
<td>ROUTERERR_FRAGMENTSBOXFULL</td>
<td>The mailbox has reached the maximum number for fragmented messages.</td>
</tr>
<tr>
<td>0x50C</td>
<td>1292</td>
<td>0x9811 050C</td>
<td>ROUTERERR_FRAGMENTTIMEOUT</td>
<td>A fragment timeout has occurred.</td>
</tr>
<tr>
<td>0x50D</td>
<td>1293</td>
<td>0x9811 050D</td>
<td>ROUTERERR_TOBEREMOVED</td>
<td>The port is removed.</td>
</tr>
</tbody>
</table>
General ADS error codes
<table>
<thead>
<tr>
<th>Hex Code</th>
<th>Decimal Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700</td>
<td>1792</td>
<td>General device error.</td>
</tr>
<tr>
<td>0x701</td>
<td>1793</td>
<td>Service is not supported by the server.</td>
</tr>
<tr>
<td>0x702</td>
<td>1794</td>
<td>Invalid index group.</td>
</tr>
<tr>
<td>0x703</td>
<td>1795</td>
<td>Invalid index offset.</td>
</tr>
<tr>
<td>0x704</td>
<td>1796</td>
<td>Reading or writing not permitted.</td>
</tr>
<tr>
<td>0x705</td>
<td>1797</td>
<td>Parameter size not correct.</td>
</tr>
<tr>
<td>0x706</td>
<td>1798</td>
<td>Invalid data values.</td>
</tr>
<tr>
<td>0x707</td>
<td>1799</td>
<td>Device is not ready to operate.</td>
</tr>
<tr>
<td>0x708</td>
<td>1800</td>
<td>Device is busy.</td>
</tr>
<tr>
<td>0x709</td>
<td>1801</td>
<td>Invalid operating system context. This can result from use of ADs function blocks in different tasks. It may be possible to resolve this through multi-task data access synchronization in the PLC.</td>
</tr>
<tr>
<td>0x70A</td>
<td>1802</td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td>0x70B</td>
<td>1803</td>
<td>Invalid parameter values.</td>
</tr>
<tr>
<td>0x70C</td>
<td>1804</td>
<td>Not found (files, etc.).</td>
</tr>
<tr>
<td>0x70D</td>
<td>1805</td>
<td>Syntax error in file or command.</td>
</tr>
<tr>
<td>0x70E</td>
<td>1806</td>
<td>Objects do not match.</td>
</tr>
<tr>
<td>0x70F</td>
<td>1807</td>
<td>Object already exists.</td>
</tr>
<tr>
<td>0x710</td>
<td>1808</td>
<td>Symbol not found.</td>
</tr>
<tr>
<td>0x711</td>
<td>1809</td>
<td>Invalid symbol version. This can occur due to an online change. Create a new handle.</td>
</tr>
<tr>
<td>0x712</td>
<td>1810</td>
<td>Device (server) is in invalid state.</td>
</tr>
<tr>
<td>0x713</td>
<td>1811</td>
<td>AdsTransMode not supported.</td>
</tr>
<tr>
<td>0x714</td>
<td>1812</td>
<td>Notification handle is invalid.</td>
</tr>
<tr>
<td>0x715</td>
<td>1813</td>
<td>Notification client not registered.</td>
</tr>
<tr>
<td>0x716</td>
<td>1814</td>
<td>No further handle available.</td>
</tr>
<tr>
<td>0x717</td>
<td>1815</td>
<td>Notification size too large.</td>
</tr>
<tr>
<td>0x718</td>
<td>1816</td>
<td>Device not initialized.</td>
</tr>
<tr>
<td>0x719</td>
<td>1817</td>
<td>Device has a timeout.</td>
</tr>
<tr>
<td>0x71A</td>
<td>1818</td>
<td>Interface query failed.</td>
</tr>
<tr>
<td>0x71B</td>
<td>1819</td>
<td>Wrong interface requested.</td>
</tr>
<tr>
<td>0x71C</td>
<td>1820</td>
<td>Class ID is invalid.</td>
</tr>
<tr>
<td>0x71D</td>
<td>1821</td>
<td>Object ID is invalid.</td>
</tr>
<tr>
<td>0x71E</td>
<td>1822</td>
<td>Request pending.</td>
</tr>
<tr>
<td>0x71F</td>
<td>1823</td>
<td>Request is aborted.</td>
</tr>
<tr>
<td>0x720</td>
<td>1824</td>
<td>Signal warning.</td>
</tr>
<tr>
<td>0x721</td>
<td>1825</td>
<td>Invalid array index.</td>
</tr>
<tr>
<td>0x722</td>
<td>1826</td>
<td>Symbol not active.</td>
</tr>
<tr>
<td>0x723</td>
<td>1827</td>
<td>Access denied.</td>
</tr>
<tr>
<td>0x724</td>
<td>1828</td>
<td>Missing license.</td>
</tr>
<tr>
<td>0x725</td>
<td>1829</td>
<td>License expired.</td>
</tr>
<tr>
<td>0x726</td>
<td>1830</td>
<td>License exceeded.</td>
</tr>
<tr>
<td>0x727</td>
<td>1831</td>
<td>Invalid license.</td>
</tr>
<tr>
<td>0x728</td>
<td>1832</td>
<td>License problem: System ID is invalid.</td>
</tr>
<tr>
<td>0x729</td>
<td>1833</td>
<td>License not limited in time.</td>
</tr>
<tr>
<td>0x72A</td>
<td>1834</td>
<td>License problem: Time in the future.</td>
</tr>
<tr>
<td>0x72B</td>
<td>1835</td>
<td>License period too long.</td>
</tr>
<tr>
<td>0x72C</td>
<td>1836</td>
<td>Exception at system startup.</td>
</tr>
<tr>
<td>0x72D</td>
<td>1837</td>
<td>License file read twice.</td>
</tr>
<tr>
<td>0x72E</td>
<td>1838</td>
<td>Invalid signature.</td>
</tr>
<tr>
<td>0x72F</td>
<td>1839</td>
<td>Invalid certificate.</td>
</tr>
<tr>
<td>0x730</td>
<td>1840</td>
<td>Public key not known from OEM.</td>
</tr>
<tr>
<td>0x731</td>
<td>1841</td>
<td>License not valid for this system ID.</td>
</tr>
<tr>
<td>0x732</td>
<td>1842</td>
<td>Demo license prohibited.</td>
</tr>
<tr>
<td>0x733</td>
<td>1843</td>
<td>Invalid function ID.</td>
</tr>
<tr>
<td>0x734</td>
<td>1844</td>
<td>Outside the valid range.</td>
</tr>
<tr>
<td>0x735</td>
<td>1845</td>
<td>Invalid alignment.</td>
</tr>
</tbody>
</table>
### TCP Winsock error codes

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x274C</td>
<td>10060</td>
<td>WSAETIMEOUT</td>
<td>A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.</td>
</tr>
<tr>
<td>0x274D</td>
<td>10061</td>
<td>WSAECONNREFUSED</td>
<td>Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.</td>
</tr>
<tr>
<td>0x2751</td>
<td>10065</td>
<td>WSAEHOSTUNREACH</td>
<td>No route to host - a socket operation referred to an unavailable host.</td>
</tr>
</tbody>
</table>

More Winsock error codes: Win32 error codes
### 8.1.2.2 TwinCAT Database Server error codes overview

<table>
<thead>
<tr>
<th>Code (hex)</th>
<th>Code (Dec)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001 + ADS error code</td>
<td>65537 - 131071</td>
<td>ADS error code from declared ADS device</td>
</tr>
<tr>
<td>0x00020001</td>
<td>131073</td>
<td>Microsoft SQL Compact database (error code)</td>
</tr>
<tr>
<td>0x00040001</td>
<td>262145</td>
<td>Microsoft SQL database (error code)</td>
</tr>
<tr>
<td>0x00080001</td>
<td>524289</td>
<td>Microsoft Access database (error code)</td>
</tr>
<tr>
<td>0x00100001</td>
<td>1048577</td>
<td>MySQL database (error code)</td>
</tr>
<tr>
<td>0x00200001</td>
<td>2097153</td>
<td>Oracle database (error code)</td>
</tr>
<tr>
<td>0x00400001</td>
<td>4194305</td>
<td>DB2 database (error code)</td>
</tr>
<tr>
<td>0x00800001</td>
<td>8388609</td>
<td>PostgreSQL database (error code)</td>
</tr>
<tr>
<td>0x01000001</td>
<td>16777217</td>
<td>Interbase/Firebird database (error code)</td>
</tr>
<tr>
<td>0x02000001</td>
<td>33554433</td>
<td>TwinCAT Database Server error code [405]</td>
</tr>
<tr>
<td>0x04000001</td>
<td>67108865</td>
<td>XML database (error code)</td>
</tr>
<tr>
<td>0x08000001</td>
<td>134217729</td>
<td>ASCII database (error code)</td>
</tr>
</tbody>
</table>

If one of the error codes mentioned above is issued at the "nErrID" output of a function block, an error has occurred during execution of an SQL statement. The SQL error code is then issued at the "sSQLState" output of the function block. The "sSQLState" output has the data type `ST_DBSQLError` [320]. For each database type individual error codes are output.


<table>
<thead>
<tr>
<th>Database type</th>
<th>Error code reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft SQL database</td>
<td>[OleDb_Errorcodes.htm][400]</td>
</tr>
<tr>
<td>Microsoft Access database</td>
<td>[OleDb_Errorcodes.htm][400]</td>
</tr>
<tr>
<td>Oracle database</td>
<td><a href="http://www.ora-code.com">http://www.ora-code.com</a></td>
</tr>
<tr>
<td>PostgreSQL database</td>
<td><a href="http://www.postgresql.org/docs/current/static/errcodes-appendix.html">http://www.postgresql.org/docs/current/static/errcodes-appendix.html</a></td>
</tr>
<tr>
<td>XML database</td>
<td>[TcDBServer_XML_Errorcodes.htm][405]</td>
</tr>
<tr>
<td>ASCII database</td>
<td>[TcDBServer_ASCII_Errorcodes.htm][405]</td>
</tr>
</tbody>
</table>
8.1.2.3 OleDB error codes
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80040E00</td>
<td>The accessor is invalid.</td>
</tr>
<tr>
<td>0x80040E01</td>
<td>It was not possible to insert a row into the row set, because the maximum number of active rows for the provider would have been exceeded.</td>
</tr>
<tr>
<td>0x80040E02</td>
<td>The accessor is write-protected. The procedure has failed.</td>
</tr>
<tr>
<td>0x80040E03</td>
<td>Values violate the database schema.</td>
</tr>
<tr>
<td>0x80040E04</td>
<td>The row handle is invalid.</td>
</tr>
<tr>
<td>0x80040E05</td>
<td>The object was open.</td>
</tr>
<tr>
<td>0x80040E06</td>
<td>Invalid chapter</td>
</tr>
<tr>
<td>0x80040E07</td>
<td>A literal value in the command could not be converted to the correct type for a reason other than data overflow.</td>
</tr>
<tr>
<td>0x80040E08</td>
<td>Invalid binding information</td>
</tr>
<tr>
<td>0x80040E09</td>
<td>Permission denied</td>
</tr>
<tr>
<td>0x80040E0A</td>
<td>The specified column contains no bookmarks or chapters.</td>
</tr>
<tr>
<td>0x80040E0B</td>
<td>Some cost limitations were rejected.</td>
</tr>
<tr>
<td>0x80040E0C</td>
<td>No command was specified for the command object.</td>
</tr>
<tr>
<td>0x80040E0D</td>
<td>No query plan was found within the specified cost limitation.</td>
</tr>
<tr>
<td>0x80040E0E</td>
<td>Invalid bookmark</td>
</tr>
<tr>
<td>0x80040E0F</td>
<td>Invalid lock mode</td>
</tr>
<tr>
<td>0x80040E10</td>
<td>No value was specified for at least one of the required parameters.</td>
</tr>
<tr>
<td>0x80040E11</td>
<td>Invalid column ID</td>
</tr>
<tr>
<td>0x80040E12</td>
<td>Invalid quota</td>
</tr>
<tr>
<td>0x80040E13</td>
<td>Invalid value</td>
</tr>
<tr>
<td>0x80040E14</td>
<td>The command contained at least one error.</td>
</tr>
<tr>
<td>0x80040E15</td>
<td>The currently executed command cannot be aborted.</td>
</tr>
<tr>
<td>0x80040E16</td>
<td>The provider offers no support for the specified dialect.</td>
</tr>
<tr>
<td>0x80040E17</td>
<td>A data source with the specified name already exists.</td>
</tr>
<tr>
<td>0x80040E18</td>
<td>The row set was created via a live datastream and cannot be restarted.</td>
</tr>
<tr>
<td>0x80040E19</td>
<td>In the current range no key matches the described characteristics.</td>
</tr>
<tr>
<td>0x80040E1B</td>
<td>The provider is unable to determine the identity for the newly added rows.</td>
</tr>
<tr>
<td>0x80040E1A</td>
<td>The ownership of this structure was transferred to the provider.</td>
</tr>
<tr>
<td>0x80040E1C</td>
<td>Non-zero weighting values are not supported as target information. The target was therefore rejected. The current target was not changed.</td>
</tr>
<tr>
<td>0x80040E1D</td>
<td>The requested conversion is not supported.</td>
</tr>
<tr>
<td>0x80040E1E</td>
<td>RowsOffset leads to position after the end of the row set, irrespective of the specified cRows value. cRowsObtained is 0.</td>
</tr>
<tr>
<td>0x80040E20</td>
<td>The provider has called an IRowsetNotify method in the consumer and has not yet received a return from the method.</td>
</tr>
<tr>
<td>0x80040E21</td>
<td>Error</td>
</tr>
<tr>
<td>0x80040E22</td>
<td>A non-zero controlling IUnknown object was specified, and the currently created object does not support aggregation.</td>
</tr>
<tr>
<td>0x80040E23</td>
<td>The current row was deleted.</td>
</tr>
<tr>
<td>0x80040E24</td>
<td>The row set does not support backward calls.</td>
</tr>
<tr>
<td>0x80040E25</td>
<td>All HROW objects have to be released before new HROW objects can be received.</td>
</tr>
<tr>
<td>0x80040E26</td>
<td>A specified memory flag was not supported.</td>
</tr>
<tr>
<td>0x80040E27</td>
<td>The comparison operator was invalid.</td>
</tr>
<tr>
<td>0x80040E28</td>
<td>The specified status flag was neither DBCOLUMNSTATUS_OK nor DBCOLUMNSTATUS_ISNULL.</td>
</tr>
<tr>
<td>0x80040E29</td>
<td>The row set cannot be processed backwards.</td>
</tr>
<tr>
<td>0x80040E2A</td>
<td>Invalid range handle.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x80040E2B</td>
<td>The specified row set was not adjacent to the rows of the specified monitoring range, and there was no overlap.</td>
</tr>
<tr>
<td>0x80040E2C</td>
<td>A transition from ALL* to MOVE* or EXTEND* was specified.</td>
</tr>
<tr>
<td>0x80040E2D</td>
<td>The specified range is not a valid subrange of the range identified by the specified monitoring range handle.</td>
</tr>
<tr>
<td>0x80040E2E</td>
<td>The provider does not support commands with several statements.</td>
</tr>
<tr>
<td>0x80040E2F</td>
<td>A specified value violated the integrity restrictions for a column or table.</td>
</tr>
<tr>
<td>0x80040E30</td>
<td>The specified type name was not recognized.</td>
</tr>
<tr>
<td>0x80040E31</td>
<td>Execution was aborted, since no further resources were available. No results were returned.</td>
</tr>
<tr>
<td>0x80040E32</td>
<td>A command object with a command hierarchy containing at least one row set could not be cloned.</td>
</tr>
<tr>
<td>0x80040E33</td>
<td>The current structure cannot be shown as text.</td>
</tr>
<tr>
<td>0x80040E34</td>
<td>The specified index already exists.</td>
</tr>
<tr>
<td>0x80040E35</td>
<td>The specified index does not exist.</td>
</tr>
<tr>
<td>0x80040E36</td>
<td>The specified index was used.</td>
</tr>
<tr>
<td>0x80040E37</td>
<td>The specified table does not exist.</td>
</tr>
<tr>
<td>0x80040E38</td>
<td>The row set has uses fully parallelism, and the value of a column was changed since the last read operation.</td>
</tr>
<tr>
<td>0x80040E39</td>
<td>Errors were found during copying.</td>
</tr>
<tr>
<td>0x80040E3A</td>
<td>A precision statement was invalid.</td>
</tr>
<tr>
<td>0x80040E3B</td>
<td>A specified decimal value was invalid.</td>
</tr>
<tr>
<td>0x80040E3C</td>
<td>Invalid table ID.</td>
</tr>
<tr>
<td>0x80040E3D</td>
<td>A specified type was invalid.</td>
</tr>
<tr>
<td>0x80040E3E</td>
<td>A column ID occurred several times in the specification.</td>
</tr>
<tr>
<td>0x80040E3F</td>
<td>The specified table already exists.</td>
</tr>
<tr>
<td>0x80040E40</td>
<td>The specified table was used.</td>
</tr>
<tr>
<td>0x80040E41</td>
<td>The specified range schema ID was not supported.</td>
</tr>
<tr>
<td>0x80040E42</td>
<td>The specified record number is invalid.</td>
</tr>
<tr>
<td>0x80040E43</td>
<td>No matching row could be found, despite the fact that the bookmark formatting was valid.</td>
</tr>
<tr>
<td>0x80040E44</td>
<td>The value of a property was invalid.</td>
</tr>
<tr>
<td>0x80040E45</td>
<td>The row set was not subdivided into chapters.</td>
</tr>
<tr>
<td>0x80040E46</td>
<td>Invalid accessor</td>
</tr>
<tr>
<td>0x80040E47</td>
<td>Invalid memory flags</td>
</tr>
<tr>
<td>0x80040E48</td>
<td>Accessors for transfer as reference are not supported by this provider.</td>
</tr>
<tr>
<td>0x80040E49</td>
<td>NULL accessors are not supported by this provider.</td>
</tr>
<tr>
<td>0x80040E4A</td>
<td>The command was not prepared.</td>
</tr>
<tr>
<td>0x80040E4B</td>
<td>The specified accessor was not a parameter accessor.</td>
</tr>
<tr>
<td>0x80040E4C</td>
<td>The specified accessor was write-protected.</td>
</tr>
<tr>
<td>0x80040E4D</td>
<td>Error during authentication.</td>
</tr>
<tr>
<td>0x80040E4E</td>
<td>The change was aborted during the notification; no columns were modified.</td>
</tr>
<tr>
<td>0x80040E4F</td>
<td>The row set consisted of a chapter, but the chapter was not enabled.</td>
</tr>
<tr>
<td>0x80040E50</td>
<td>Invalid source handle</td>
</tr>
<tr>
<td>0x80040E51</td>
<td>The provider is unable to derive parameter information, and SetParameterInfo was not called.</td>
</tr>
<tr>
<td>0x80040E52</td>
<td>The data source object is already initialized.</td>
</tr>
<tr>
<td>0x80040E53</td>
<td>The provider does not support this method.</td>
</tr>
<tr>
<td>0x80040E54</td>
<td>The number of rows with pending modifications exceeds the specified limit.</td>
</tr>
<tr>
<td>0x80040E55</td>
<td>The specified column did not exist.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x80040E56</td>
<td>Changes are pending in a row with a reference counter of zero.</td>
</tr>
<tr>
<td>0x80040E57</td>
<td>A literal value in the command let to a range violation for the type of the assigned column.</td>
</tr>
<tr>
<td>0x80040E58</td>
<td>The transferred HRESULT value was invalid.</td>
</tr>
<tr>
<td>0x80040E59</td>
<td>The transferred LookupID value was invalid.</td>
</tr>
<tr>
<td>0x80040E5A</td>
<td>The transferred DynamicErrorID value was invalid.</td>
</tr>
<tr>
<td>0x80040E5B</td>
<td>No visible data for a newly added row that has not yet been updated can be retrieved.</td>
</tr>
<tr>
<td>0x80040E5C</td>
<td>Invalid conversion flag</td>
</tr>
<tr>
<td>0x80040E5D</td>
<td>The specified parameter name was not recognized.</td>
</tr>
<tr>
<td>0x80040E5E</td>
<td>Several memory objects cannot be open simultaneously.</td>
</tr>
<tr>
<td>0x80040E5F</td>
<td>The requested filter could not be opened.</td>
</tr>
<tr>
<td>0x80040E60</td>
<td>The requested sequence could not be opened.</td>
</tr>
<tr>
<td>0x80040E65</td>
<td>The transferred columnID value was invalid.</td>
</tr>
<tr>
<td>0x80040E67</td>
<td>The transferred command has no DBID value.</td>
</tr>
<tr>
<td>0x80040E68</td>
<td>The transferred DBID value already exists.</td>
</tr>
<tr>
<td>0x80040E69</td>
<td>The maximum number of session objects supported by this provider has already been reached. The consumer must release at least one current session object, before a new session object can be retrieved.</td>
</tr>
<tr>
<td>0x80040E72</td>
<td>The index ID is invalid.</td>
</tr>
<tr>
<td>0x80040E73</td>
<td>The specified initialization character sequence does not match the specification.</td>
</tr>
<tr>
<td>0x80040E74</td>
<td>The OLE DB master enumerator has not returned any providers that match the requested SOURCES_TYPE value.</td>
</tr>
<tr>
<td>0x80040E75</td>
<td>The initialization character sequence indicates a provider that does not match the currently active provider.</td>
</tr>
<tr>
<td>0x80040E76</td>
<td>The specified DBID value is invalid.</td>
</tr>
<tr>
<td>0x80040E6A</td>
<td>Invalid value for trust recipient.</td>
</tr>
<tr>
<td>0x80040E6B</td>
<td>The trust recipient is not intended for the current data source.</td>
</tr>
<tr>
<td>0x80040E6C</td>
<td>The trust recipient offers no support for memberships/list.</td>
</tr>
<tr>
<td>0x80040E6D</td>
<td>The object is invalid, or the provider is unknown.</td>
</tr>
<tr>
<td>0x80040E6E</td>
<td>The object has no owner.</td>
</tr>
<tr>
<td>0x80040E6F</td>
<td>The transferred access entry list is invalid.</td>
</tr>
<tr>
<td>0x80040E70</td>
<td>The trust recipient transferred as owner is invalid, or the provider is unknown.</td>
</tr>
<tr>
<td>0x80040E71</td>
<td>The permission transferred in the access entry list is invalid.</td>
</tr>
<tr>
<td>0x80040E77</td>
<td>The ConstraintType value was invalid or was not supported by the provider.</td>
</tr>
<tr>
<td>0x80040E78</td>
<td>The ConstraintType value was not DBCONSTRAINTTYPE_FOREIGNKEY, and cForeignKeyColumns was not zero.</td>
</tr>
<tr>
<td>0x80040E79</td>
<td>The Deferability value was invalid or was not supported by the provider.</td>
</tr>
<tr>
<td>0x80040E80</td>
<td>The MatchType value was invalid or was not supported by the provider.</td>
</tr>
<tr>
<td>0x80040E8A</td>
<td>The UpdateRule or DeleteRule value was invalid or was not supported by the provider.</td>
</tr>
<tr>
<td>0x80040E8B</td>
<td>Invalid restriction ID.</td>
</tr>
<tr>
<td>0x80040E8C</td>
<td>The dwFlags value was invalid.</td>
</tr>
<tr>
<td>0x80040E8D</td>
<td>The rguidColumnType value pointed to a GUID that does not match the object type of this column, or this column was not specified.</td>
</tr>
<tr>
<td>0x80040E91</td>
<td>No source row exists.</td>
</tr>
<tr>
<td>0x80040E92</td>
<td>The OLE DB object represented by this URL is locked by at least one other process.</td>
</tr>
<tr>
<td>0x80040E93</td>
<td>The client requested an object type that is only for lists.</td>
</tr>
<tr>
<td>0x80040E94</td>
<td>The calling process requested write access for a write-protected object.</td>
</tr>
<tr>
<td>0x80040E95</td>
<td>The provider was unable to establish a connection with the server for this object.</td>
</tr>
<tr>
<td>0x80040E96</td>
<td>The provider was unable to establish a connection with the server for this object.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x80040E97</td>
<td>Timeout during binding to the object</td>
</tr>
<tr>
<td>0x80040E98</td>
<td>The provider was unable to create an object with this URL, since an object named by</td>
</tr>
<tr>
<td></td>
<td>this URL already exists.</td>
</tr>
<tr>
<td>0x80040E8E</td>
<td>The requested URL was outside the valid range.</td>
</tr>
<tr>
<td>0x80040E90</td>
<td>The column or restriction could not be deleted, since a dependent view or restriction</td>
</tr>
<tr>
<td></td>
<td>refers to it.</td>
</tr>
<tr>
<td>0x80040E99</td>
<td>The restriction already exists.</td>
</tr>
<tr>
<td>0x80040E9A</td>
<td>The object cannot be created with this URL, since the server has insufficient physical</td>
</tr>
<tr>
<td></td>
<td>memory.</td>
</tr>
<tr>
<td>0x00040EC0</td>
<td>During retrieval of the requested number of rows the total number of active rows</td>
</tr>
<tr>
<td></td>
<td>supported by this row set was exceeded.</td>
</tr>
<tr>
<td>0x00040EC1</td>
<td>At least one column type is not compatible; conversion errors may occur during</td>
</tr>
<tr>
<td></td>
<td>copying.</td>
</tr>
<tr>
<td>0x00040EC2</td>
<td>Information on the parameter type were disabled by the calling process.</td>
</tr>
<tr>
<td>0x00040EC3</td>
<td>The bookmark for a deleted or irrelevant row was skipped.</td>
</tr>
<tr>
<td>0x00040EC5</td>
<td>No further row sets are available.</td>
</tr>
<tr>
<td>0x00040EC6</td>
<td>Start or end of the row set or chapter reached.</td>
</tr>
<tr>
<td>0x00040EC7</td>
<td>The command was executed again by the provider.</td>
</tr>
<tr>
<td>0x00040EC8</td>
<td>The data buffer for the variable is full.</td>
</tr>
<tr>
<td>0x00040EC9</td>
<td>No further results are available.</td>
</tr>
<tr>
<td>0x00040ECA</td>
<td>The server is unable to cancel or downgrade a lockout until a transaction is complete.</td>
</tr>
<tr>
<td>0x00040ECB</td>
<td>The specified weighting value was not supported or exceeded the supported limit. The value</td>
</tr>
<tr>
<td></td>
<td>was set to 0 or to the limit.</td>
</tr>
<tr>
<td>0x00040ECC</td>
<td>For this reason the consumer rejects further notification calls.</td>
</tr>
<tr>
<td>0x00040ECD</td>
<td>The input dialect was ignored, and the text was returned in another dialect.</td>
</tr>
<tr>
<td>0x00040ECE</td>
<td>The consumer rejects further notification calls for this phase.</td>
</tr>
<tr>
<td>0x00040ECF</td>
<td>For this reason the consumer rejects further notification calls.</td>
</tr>
<tr>
<td>0x00040ED0</td>
<td>The operation is processed asynchronously.</td>
</tr>
<tr>
<td>0x00040ED1</td>
<td>To reach the start of the row set, the provider has to execute the query again. Either</td>
</tr>
<tr>
<td></td>
<td>the order of the columns has changed, or columns were added to the row set, or columns</td>
</tr>
<tr>
<td></td>
<td>were removed from the row set.</td>
</tr>
<tr>
<td>0x00040ED2</td>
<td>The method had several errors. The errors were returned in the error array.</td>
</tr>
<tr>
<td>0x00040ED3</td>
<td>Invalid row handle</td>
</tr>
<tr>
<td>0x00040ED4</td>
<td>A specified HROW object referred to a permanently deleted row.</td>
</tr>
<tr>
<td>0x00040ED5</td>
<td>The provider was unable to trace all modifications. The client has to retrieve the data</td>
</tr>
<tr>
<td></td>
<td>assigned the monitoring range again via another method.</td>
</tr>
<tr>
<td>0x00040ED6</td>
<td>The execution was terminated because no more resources were available. The results</td>
</tr>
<tr>
<td></td>
<td>received up to this time were returned, but the execution cannot continue.</td>
</tr>
<tr>
<td>0x00040ED8</td>
<td>A lockout was upgraded relative to the specified value.</td>
</tr>
<tr>
<td>0x00040ED9</td>
<td>At least one property was changed according to the options permitted by the provider.</td>
</tr>
<tr>
<td>0x00040EDA</td>
<td>Error</td>
</tr>
<tr>
<td>0x00040EDB</td>
<td>A specified parameter was invalid.</td>
</tr>
<tr>
<td>0x00040EDC</td>
<td>Due to the update of this row several rows in the data source had to be updated.</td>
</tr>
<tr>
<td>0x00040ED7</td>
<td>The binding failed, since the provider was not able to meet all binding flags or</td>
</tr>
<tr>
<td></td>
<td>properties.</td>
</tr>
<tr>
<td>0x00040EDD</td>
<td>The row contains no row-specific columns.</td>
</tr>
</tbody>
</table>
8.1.2.4 ASCII error codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Function not available</td>
</tr>
<tr>
<td>2</td>
<td>Syntax error</td>
</tr>
<tr>
<td>3</td>
<td>File could not be opened.</td>
</tr>
</tbody>
</table>

8.1.2.5 XML error codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Function not available</td>
</tr>
<tr>
<td>2</td>
<td>XML file could not be loaded</td>
</tr>
<tr>
<td>3</td>
<td>XML schema could not be loaded</td>
</tr>
<tr>
<td>4</td>
<td>Syntax error</td>
</tr>
<tr>
<td>5</td>
<td>Table could not be created</td>
</tr>
<tr>
<td>6</td>
<td>The INSERT VALUE list does not match the columns list</td>
</tr>
<tr>
<td>7</td>
<td>PLC structure is not large enough</td>
</tr>
<tr>
<td>8</td>
<td>XML file could not be created</td>
</tr>
<tr>
<td>9</td>
<td>XML database not found.</td>
</tr>
<tr>
<td>10</td>
<td>XML table not found</td>
</tr>
</tbody>
</table>

8.1.2.6 Internal error codes

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x02000001</td>
<td>NULL values are not allowed</td>
</tr>
<tr>
<td>0x02000002</td>
<td>FB_DBRead selected value is NULL</td>
</tr>
<tr>
<td>0x02000003</td>
<td>DBID is unknown</td>
</tr>
<tr>
<td>0x02000004</td>
<td>ADSDevID is unknown</td>
</tr>
<tr>
<td>0x02000005</td>
<td>No open database connection found for DBID xy</td>
</tr>
<tr>
<td>0x02000006</td>
<td>No open ADS device connection found for ADSDevID xy</td>
</tr>
</tbody>
</table>

8.2 FAQ - frequently asked questions and answers

In this section frequently asked questions are answered, in order to facilitate your work with the TwinCAT Database Server.

If you have any further questions, contact please our Support.

1. What performance can be achieved with the TwinCAT Database Server? [406]

2. Are so-called stored procedures supported? [406]

3. Which database types does the TwinCAT Database Server support? [406]

4. Can old database server configurations still be used in later versions? [406]

5. How can individual variables be written into an existing database structure or read from it? [406]

6. Can several records be written into a database simultaneously? [406]

7. How can the TwinCAT Database Server be operated in a network? [406]
8. Which functions of the TwinCAT Database Server are supported by the database type "XML"? [► 406]

9. Which Visual Studio versions are currently supported by the database server configurator? [► 406]

What performance can be achieved with the TwinCAT Database Server?
This question cannot be answered in general terms. The performance that can be achieved depends on the hardware used, the actions to be execute (e.g. ring buffer logging) and the number of variables. Another key factor is the database type that is used.

Are so-called stored procedures supported?
Yes, the TwinCAT Database Server supports stored procedures. The function block FB_SQLStoredProcedure [► 198] is provided for this purpose in the PLC library. The SQL Query Editor [► 41] can be used to test stored procedures, and corresponding PLC code for the function block FB_SQLStoredProcedure can be generated. Not all databases support this function.

Which database types does the TwinCAT Database Server support?
Information on the supported databases can be found in section "Databases [► 122]".

Can old database server configurations still be used in later versions?
It goes without saying that we aim to ensure compatibility. This was also taken into account during major version upgrades or complete redesigns (e.g. old: 3.0.x, new: 3.1.x). Further details can be found in section "Compatibility [► 19]".

How can individual variables be written into an existing database structure or read from it?
The function block FB_SQLCommand [► 192] can be used to write individual variables into an existing database structure or read from it.

Can several records be written into a database simultaneously?
This depends on the database used. With a Microsoft SQL database this would be possible in conjunction with the function block FB_SQLCommand [► 192], since several SQL insert commands (separated by semicolon) can be transferred to the PLC function block.

How can the TwinCAT Database Server be operated in a network?
The TwinCAT Database Server can be used in a network in several ways. Further information on support network topologies can be found in section "Areas of application and network topologies [► 17]".

Which functions of the TwinCAT Database Server are supported by the database type "XML"?
The "XML" database type supports the full functionality of the TwinCAT Database Server, except for "stored procedures". The XML file can be used to communicate with any other database via SQL commands, and PLC values can be logged in the XML file with the cyclic write mode. In addition, it is possible to execute XPath commands and read the corresponding XML tags. Further information can be found in section "XML database [► 132]".

Which Visual Studio versions are currently supported by the database server configurator?
Currently the Visual Studio® versions 2013, 2015 and 2017 are supported with our configurator integration [► 22].

8.3 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

Beckhoff’s branch offices and representatives

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages: https://www.beckhoff.com

You will also find further documentation for Beckhoff components there.
Beckhoff Support

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

Hotline: +49 5246 963 157
Fax: +49 5246 963 9157
e-mail: support@beckhoff.com

Beckhoff Service

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

Hotline: +49 5246 963 460
Fax: +49 5246 963 479
e-mail: service@beckhoff.com

Beckhoff Headquarters

Beckhoff Automation GmbH & Co. KG
Huelshorstweg 20
33415 Verl
Germany
Phone: +49 5246 963 0
Fax: +49 5246 963 198
e-mail: info@beckhoff.com
web: https://www.beckhoff.com