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 PLC libraries ............................................................................................................................................. 16
   4.1 Overview .......................................................................................................................................... 16
   4.2 Function blocks ................................................................................................................................ 16
      4.2.1 FB_XmlSrvRead ......................................................................................................................... 17
      4.2.2 FB_XmlSrvWrite ....................................................................................................................... 18
      4.2.3 FB_XmlSrvReadByName .......................................................................................................... 19
      4.2.4 FB_XmlSrvWriteByName ......................................................................................................... 20
   4.3 Global constants ............................................................................................................................... 21
      4.3.1 Global Variables ....................................................................................................................... 21
      4.3.2 Library version ......................................................................................................................... 22
5 Examples .................................................................................................................................................. 23
   5.1 Getting Started ................................................................................................................................ 23
   5.2 Function blocks ................................................................................................................................ 24
   5.3 Further Samples ............................................................................................................................... 27
   5.4 Production example ........................................................................................................................... 29
6 Appendix .................................................................................................................................................. 32
   6.1 Overview of TC3 XML Server error codes....................................................................................... 32
   6.2 Internal error codes of the TwinCAT 3 XML Server .......................................................................... 32
   6.3 FAQ - Frequently asked questions and their answers ....................................................................... 32
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®, 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>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚨 DANGER</td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td>🚨 WARNING</td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td>🚨 CAUTION</td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td>🚨 NOTE</td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

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

The TwinCAT XML Server provides a PLC library wherewith write-/read-access for XML data can be realized. The XML Server is characteristic for its easy handling. It is especially suitable for loading initialization data, as it is often needed to start up a machine.

Besides easy initializing from an XML file, the XML Server allows formatted saving of PLC variables in an XML file. The structure of a variable in the XML document matches the structure of the variable in the PLC. This gives you direct access to individual subelements of a variable in the XML file. Only subelements (elements of a structure or an array), which are defined in the XML file will be transferred. When the PLC variables are written, missing elements can optionally be added.

Components

- TwinCAT XML Server: A service that starts along with TwinCAT. It is the connection between TwinCAT and the XML file.
- PLC library: The PLC library offers four function blocks for read and write operations. They allow formatted saving of the PLC variables in an XML file and initializing TwinCAT variables with data from an XML file.

Functional principle

The XML Server communicates with the TwinCAT PLC via ADS. During a writing process the variables will be converted into text and stored in the XML file through the MSXML DOM Parser. The XML file does not include any information about datatypes, but just the name of the variable and its value: <name_of_variable>value</name_of_variable>

During a reading process the datatype will be determined for each variable via ADS. The corresponding XML text will be converted properly.
3 Installation

3.1 System Requirements

<table>
<thead>
<tr>
<th>Technical data</th>
<th>TF6421 XML Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target system</td>
<td>Windows XP, Windows 7/8, Windows CE PC (x86-compatible), ARM</td>
</tr>
<tr>
<td>Min. TwinCAT version</td>
<td>3.1 Build 4011</td>
</tr>
<tr>
<td>Min. TwinCAT-Level</td>
<td>TwinCAT PLC</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.

![Dialog box for installation](image)

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

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

![Confirmation dialog box](image)
7. Select Finish to exit the setup.

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

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.

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

6. Open the Order Information (Runtime) tab.
   - In the tabular overview of licenses, the previously selected license is displayed with the status "missing".
7. Click **7-Day Trial License...** to activate the 7-day trial license.

![License Activation](image)

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

![Enter Security Code](image)

- Enter the code exactly as it is displayed and confirm the entry.

- 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.

8. 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 [► 8]** section.
After the installation, the installation folder contains three directories (one directory per hardware platform):

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

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

Example: "TF6310" installation folder

![Image showing 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`

**Software upgrade**

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

1. Open the CE Explorer by clicking **Start > Run** and entering "Explorer".
2. Navigate to `\Hard Disk\TwinCAT\Functions\TFxxxx\xxxx`.
3. Rename the file `Tc*.exe` to `Tc*.old`.
4. Restart the Windows CE device.
5. Transfer the new CAB file to the Windows CE device.
6. Run the CAB file on the Windows CE device and install the new version.
7. Delete the file *Tc*.old.
8. Restart the Windows CE device.
⇒ The new version is active after the restart.
4 PLC libraries

4.1 Overview

The PLC library **Tc2_TcXmlDataSrv** is supplied with the TC3 XML Server and copied into folder `C:\TwinCAT\3.1\Components\Plc\Managed Libraries\Beckhoff Automation GmbH` during installation.

There are two function blocks for reading variables from the XML file:
- FB_XmlSrvRead
- FB_XmlSrvReadByName

and two function blocks for writing PLC variables to the XML file:
- FB_XmlSrvWrite
- FB_XmlSrvWriteByName

The first version (FB_XMLSrvRead, FB_XMLSrvWrite) uses the address and the size of the PLC variable for specifying the variable. The second version (FB_XMLSrvReadByName, FB_XMLSrvWriteByName) uses the symbol name for specifying the variable. The first version offers higher performance. In addition, the path of the XML file and the location of the variables within the XML document has to be transferred as input parameter to the function blocks.

**Primitive data types**

The following primitive data types are supported:

<table>
<thead>
<tr>
<th>Data type</th>
<th>PLC example</th>
<th>XML example</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDINT, DINT, UINT, INT, USINT, SINT, DWORD, WORD, BYTE</td>
<td>value1 : DINT := -1; value2 : UDINT := 65535;</td>
<td><code>&lt;dataentry&gt; &lt;MAIN.value1&gt;-1&lt;/MAIN.value1&gt; &lt;MAIN.value2&gt;65535&lt;/MAIN.value2&gt; &lt;/dataentry&gt;</code></td>
</tr>
<tr>
<td>LREAL, REAL</td>
<td>value1 : LREAL = 1.2;</td>
<td><code>&lt;dataentry&gt; &lt;MAIN.value1&gt;1.2&lt;/MAIN.value1&gt; &lt;/dataentry&gt;</code></td>
</tr>
<tr>
<td>STRING</td>
<td>str1 : STRING = 'hallo';</td>
<td><code>&lt;dataentry&gt; &lt;MAIN.str1&gt;hallo&lt;/MAIN.str1&gt; &lt;/dataentry&gt;</code></td>
</tr>
<tr>
<td>TIME, DATE, TOD, DT</td>
<td>date1:DATE :=D#2005-05-04; (* Time types are stored in the XML file as DWORD*)</td>
<td><code>&lt;dataentry&gt; &lt;MAIN.date1&gt;1115164800&lt;/MAIN.date1&gt; &lt;/dataentry&gt;</code></td>
</tr>
<tr>
<td>BOOL</td>
<td>bool1:BOOL := TRUE; bool2:BOOL := FALSE;</td>
<td><code>&lt;dataentry&gt; &lt;MAIN.bool1&gt;true&lt;/MAIN.bool1&gt; &lt;MAIN.bool2&gt;false&lt;/MAIN.bool2&gt; &lt;/dataentry&gt;</code></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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>
4.2 Function blocks

4.2.1 FB_XmlSrvRead

The function block FB_XmlSrvRead can be used to initialise a PLC variable with data from an XML file. The input variable sXPath has to point to a valid node in the XML file specified via sFilePath. The symbol to be initialised is identified unambiguously by the symbol address and size.

### VAR_INPUT

```plaintext
VAR_INPUT
  sNetId : T_AmsNetId;
  ePath  : E_OpenPath := PATH_GENERIC;
  nMode  : WORD := XMLSRV_SKIPMISSING;
  pSymAddr : DWORD;
  cbSymSize : UDINT;
  sFilePath : T_MaxString;
  sXPath  : T_MaxString;
  bExecute : BOOL;
  tTimeout : TIME := T#60s;
END_VAR
```

- **sNetId**: String containing the network address of the TwinCAT 3 XML server. For the local computer (default) an empty string may be specified.
- **ePath**: This input can be used to select a TwinCAT system path on the target device for opening the file.
- **nMode**: This input can be used to control how the XML file is evaluated. Only the mode XMLSRV_SKIPMISSING is initially supported for the command XmlSrvRead.
- **pSymAddr**: Address of the PLC variable to which the data from the XML file are to be written.
- **cbSymSize**: Size of the PLC variable to which the data from the XML file are to be written.
- **sFilePath**: Contains the path and file name for the file to be opened. The path can only point to the local computer’s file system! This means that network paths cannot be used here!
- **sXPath**: Contains the address of the tag in the XML document from which the data are to be written. The address must be a valid XPath instruction. The name of the tag must be different from the name of the symbol.
- **bExecute**: The block is activated by a positive edge at this input.
- **tTimeout**: Maximum time allowed for the execution of the function block.

### VAR_OUTPUT

```plaintext
VAR_OUTPUT
  bBusy : BOOL;
  bError : BOOL;
  nErrId : UDINT;
END_VAR
```

- **bBusy**: This output is set when the function block is activated. It remains set until feedback is received.
**bError**: This output is set up after the bBusy output has been reset if there has been an error in transmission of the command.

**nErrId**: Returns the TC3 XML server error number [32], if a 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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>

### 4.2.2 FB_XmlSrvWrite

<table>
<thead>
<tr>
<th>FB_XMLSRVWRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetId: T_AmsNetId</td>
</tr>
<tr>
<td>ePath: E_OpenPath</td>
</tr>
<tr>
<td>nMode: WORD</td>
</tr>
<tr>
<td>pSymAddr: DWORD</td>
</tr>
<tr>
<td>cbSymSize: UDINT</td>
</tr>
<tr>
<td>sFilePath: T_MaxString</td>
</tr>
<tr>
<td>sXPath: T_MaxString</td>
</tr>
<tr>
<td>bExecute: BOOL</td>
</tr>
<tr>
<td>tTimeout: TIME</td>
</tr>
</tbody>
</table>

The function block FB_XmlSrvWrite can be used for writing the value of a PLC variable into an XML file. The input variable sXPath has to point to a valid node in the XML file specified via sFilePath. The symbol to be written is identified unambiguously by the symbol address and size.

### VAR_INPUT

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetId: T_AmsNetId</td>
</tr>
<tr>
<td>ePath: E_OpenPath := PATH_GENERIC</td>
</tr>
<tr>
<td>nMode: WORD := XMLSRV_SKIPMISSING</td>
</tr>
<tr>
<td>pSymAddr: DWORD</td>
</tr>
<tr>
<td>cbSymSize: UDINT</td>
</tr>
<tr>
<td>sFilePath: T_MaxString</td>
</tr>
<tr>
<td>sXPath: T_MaxString</td>
</tr>
<tr>
<td>bExecute: BOOL</td>
</tr>
<tr>
<td>tTimeout: TIME := T#60s</td>
</tr>
</tbody>
</table>

**sNetId**: String containing the network address of the TwinCAT 3 XML server. For the local computer (default) an empty string may be specified.

**ePath**: This input can be used to select a TwinCAT system path on the target device for opening the file.

**nMode**: This input can be used to control which data are written into the XML file. XMLSRV_SKIPMISSING and XMLSRV_ADDMISSING mode are available for the XmlSrvWrite command. In XMLSRV_SKIPMISSING mode, only those subelements of a PLC symbol that already exist in the XML file are written to the XML file. In XMLSRV_ADDMISSING mode, missing subelements are added to the XML file.

**pSymAddr**: Address of the PLC variable to be written to the XML file.

**cbSymSize**: Size of the PLC variable to be written to the XML file.

**sFilepath**: Contains the path and file name for the file to be opened. The path can only point to the local computer’s file system! This means that network paths cannot be used here!

**sXPath**: Contains the address of the tag in the XML document from which the data are to be written. The address must be a valid XPath instruction. The name of the tag must be different from the name of the symbol.

**bExecute**: The block is activated by a positive edge at this input.
**tTimeout**: Maximum time allowed for the execution of the function block.

**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**: This output is set when the function block is activated. It remains set until feedback is received.

**bError**: This output is set up after the bBusy output has been reset if there has been an error in transmission of the command.

**nErrId**: Returns the TC3 XML server error number \([32]\), if a 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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>

**4.2.3 FB_XmlSrvReadByName**

The function block FB_XmlSrvReadByName can be used to initialize a PLC variable with data from an XML file. The input variable sXPath has to point to a valid node in the XML file specified via sFilePath. The symbol to be initialized is identified unambiguously by the symbol name.

**VAR_INPUT**

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNetId : T_AmsNetId;</td>
</tr>
<tr>
<td>ePath : E_OpenPath := PATH GENERIC;</td>
</tr>
<tr>
<td>nMode : WORD := XMLSRV_SKIPMISSING;</td>
</tr>
<tr>
<td>sSymName : T_MaxString;</td>
</tr>
<tr>
<td>sFilePath : T_MaxString;</td>
</tr>
<tr>
<td>sXPath : T_MaxString;</td>
</tr>
<tr>
<td>bExecute : BOOL;</td>
</tr>
<tr>
<td>tTimeout : TIME := T#60s;</td>
</tr>
</tbody>
</table>

**sNetId**: String containing the network address of the TwinCAT 3 XML server. For the local computer (default) an empty string may be specified.

**ePath**: This input can be used to select a TwinCAT system path on the target device for opening the file.

**nMode**: This input can be used to control how the XML file is evaluated. Only the mode XMLSRV_SKIPMISSING is currently supported for the command XmlSrvReadByName.

**sSymName**: Name of the PLC symbol to which the data from the XML file are to be written.

**sFilePath**: Contains the path and file name for the file to be opened. The path can only point to the local computer’s file system! This means that network paths cannot be used here!
sXPath: Contains the address of the tag in the XML document from which the data are to be written. The address must be a valid XPath instruction. The name of the tag must be different from the name of the symbol.

bExecute: The block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

VAR_OUTPUT

VAR_OUTPUT
bBusy : BOOL;
bError : BOOL;
nErrId : UDINT;
END_VAR

bBusy: This output is set when the function block is activated. It remains set until feedback is received.

bError: This output is set up after the bBusy output has been reset if there has been an error in transmission of the command.

nErrId: Returns the TC3 XML server error number [32], if a 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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>

4.2.4 FB_XmlSrvWriteByName

The function block FB_XmlSrvWriteByName can be used for writing the value of a PLC variable into an XML file. The input variable sXPath has to point to a valid node in the XML file specified via sFilePath. The symbol to be written is identified unambiguously by the symbol name.

VAR_INPUT

VAR_INPUT
sNetId : T_AmsNetId;
ePath : E_OpenPath := PATH_GENERIC;
nMode : WORD := XMLSRV_SKIPMISSING;
sSymName : T_MaxString;
sFilePath : T_MaxString;
sXPath : T_MaxString;
bExecute : BOOL;
tTimeout : TIME := T#60s;
END_VAR

sNetId: String containing the network address of the TwinCAT 3 XML server. For the local computer (default) an empty string may be specified.

ePath: This input can be used to select a TwinCAT system path on the target device for opening the file.
nMode: This input can be used to control which data are written into the XML file. XMLSRV_SKIPMISSING and XMLSRV_ADDMISSING mode are available for the XmlSrvWriteByte command. In XMLSRV_SKIPMISSING mode, only those subelements of a PLC symbol that already exist in the XML file are written to the XML file. In XMLSRV_ADDMISSING mode, missing subelements are added to the XML file.

sSymName: Name of the PLC symbol to be written to the XML file.

sFilePath: Contains the path and file name for the file to be opened. The path can only point to the local computer’s file system! This means that network paths cannot be used here!

sXPath: Contains the address of the tag in the XML document from which the data are to be written. The address must be a valid XPath instruction. The name of the tag must be different from the name of the symbol.

bExecute: The block is activated by a positive edge at this input.

tTimeout: Maximum time allowed for the execution of the function block.

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: This output is set when the function block is activated. It remains set until feedback is received.

bError: This output is set up after the bBusy output has been reset if there has been an error in transmission of the command.

nErrId: Returns the TC3 XML server error number [32], if a 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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>

4.3 Global constants

4.3.1 Global Variables

VAR_GLOBAL CONSTANT

| XMLSRV_AMSPORT :UINT :=10600; |
| XMLSRV_IGR_CLOSE :UDINT := 121; |
| XMLSRV_IGR_READ :UDINT := 122; |
| XMLSRV_IGR_WRITE :UDINT := 123; |
| XMLSRV_IGR_OPENREAD :UDINT := 124; |
| XMLSRV_IGR_OPENWRITE :UDINT := 125; |
| XMLSRV_SKIPMISSING :WORD := 0; |
| XMLSRV_ADDMISSING :WORD := 1; (*for write commands*) |
| XMLSRV_MAX_FRAGSIZE :UDINT := 16#40000; |
| XMLSRVERROR_INTERNAL :UDINT:= 16#8000; |
| XMLSRVERROR_NOTFOUND :UDINT:= 16#8001; |
| XMLSRVERROR_PARSERROR :UDINT:= 16#8002; |
| XMLSRVERROR_INCOMPATIBLE :UDINT:= 16#8003; |
| XMLSRVERROR_NOMEMORY :UDINT:= 16#8004; |
| XMLSRVERROR_ADDNODE :UDINT:= 16#8005; |
| XMLSRVERROR_INVALIDPATH :UDINT:= 16#8006; |
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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>

### 4.3.2 Library version

All libraries have a specific version. This version is shown in the PLC library repository too.

A global constant contains the library version information:

```
VAR_GLOBAL CONSTANT
  stLibVersion_Tc2_TcXmlDataSrv : ST_LibVersion;
END_VAR
```

**Tc2_TcXmlDataSrv**: Version number of the Tc2_TcXmlDataSrv library (type: ST_LibVersion)

To compare the existing version to a required version the function F_CmpLibVersion (defined in Tc2_System library) is offered.

**Comparing versions**

All other options for comparing library versions, which you may know from TwinCAT 2, are outdated.
5 Examples

The following pages contain examples (referred to as samples) that illustrate the operation of the TC3 XML server. The examples are numbered consecutively and can be downloaded as a PLC project from here: https://infosys.beckhoff.com/content/1033/TF6421_Tc3_XML_Server/Resources/zip/1635013643.zip

- Getting started [23]
  This section describes the general operation of the TwinCAT XML Data Server, the function of structures and arrays, and the access to individual nodes.

- Function blocks [24]
  The application and configuration of the function blocks of the PLC library TcXmlDataSrv.Lib is explained based on four examples. (Examples 1-4)

- Further examples [27]
  This document contains further examples, which describe the one-time initialization during PLC startup and cyclic writing, for instance. (Examples 5-6)

- Production example [29]
  The production example illustrates the processing of a small production order. It uses FB_XmlSrvReadByName and FB_XmlSrvWriteByName. (Example 7)

5.1 Getting Started

The following section describes the basic operating principle of the TC3 XML server, based on short examples.

The global variable .Var1 of type DINT is defined in the PLC. It should be saved in an XML file as follows:

```xml
<dataentry>
  <Var1>10</Var1>
</dataentry>
```

To this end the input variables of the function block FB_XmlSrvWrite have to be set as follows:

```plaintext
fbRead.pSymAddr := ADR(value1);
fbRead.cbSymSize := SIZEOF(value1);
fbRead.sFilePath := 'C:\Test.xml'; (*Pfad zur XML-Datei*)
fbRead.sXPath := '/dataentry/Var1';
```

The root element name of the variables in the XML file is freely selectable. Simply adapt the path in the input variable sXPath. An XML file may contain definitions for several variables:

```xml
<dataentry>
  <Var1>10</Var1>
  <Var2>100</Var2>
  <Var3>
    <a>100</a>
    <b>10</b>
  </Var3>
  <Var4>10</Var4>
</dataentry>
```

To access the symbol .Var3.a, for example, sXPath must be set to 'dataentry/Var3.a'.

**Structures**

XML files have the same hierarchical structure as the PLC. Individual subelements of the XML file may be skipped: The subelements of the structure must have the same names as in the PLC, otherwise they are skipped. If subelements of the XML file cannot be converted to the correct data type, they are also skipped.

**Example:**

The global variable .Var2 of type ST_MYSTRUCT is defined in the PLC:

```plaintext
TYPE ST_MYSTRUCT:
  STRUCT
    a: UINT;
    b: DINT;
    c: LREAL;
```
Examples

```plaintext
d: STRING;
END_STRUCT
END_TYPE
```

The XML file could then look as follows:

```xml
<variables>
  <Var1>10</Var1>
  <Var2> <!-- sXPath := '/variables/Var2' -->
    <a>100</a>
    <b>-10</b>
    <c>1.2</c>
    <d>Hallo</d>
  </Var2>
</variables>
```

In this case all subelements are defined fully and correctly, so that the variable is fully initialized. In the following example, on the other hand, only subelement c is serialized:

```xml
<variables>
  <Var1>10</Var1>
  <Variable2> <!-- sXPath := '/variables/Variable2' -->
    <Info>dies ist ein Test</Info>
    <a>-100</a>
    <c>1.2</c>
  </Variable2>
</variables>
```

Subelement a cannot be converted because it is negative and UINT is required. Subelement b is missing completely. The tag `<Info>` is skipped, because it is not defined in the PLC file.

Arrays

In order to specify an array index, the attribute "index" has to be used for the individual array elements. Individual array elements can be omitted, in which case they will be skipped.

Example:

A variable `.array1` of type `ARRAY[1..4] OF DINT` is defined in the PLC. The XML file will then look as follows:

```xml
<dataentry>
  <array1 index="1">10</array1>
  <array1 index="2">10</array1>
  <array1 index="3">10</array1>
  <array1 index="4">10</array1>
</dataentry>
```

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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>

5.2 Function blocks

The following samples show the handling of the functionblocks of the Tc2_XmlDataSrv-Library. The PLC project, which contains the samples, can be downloaded here: https://infosys.beckhoff.com/content/1033/TF6421_Tc3_XML_Server/Resources/zip/1635013643.zip

All samples work with the structure `ST_MYSTRUCT`, which in turn includes `ST_INNTERSTRUCT`. Both Structures are shown in the following:

Structures

```plaintext
TYPE ST_MYSTRUCT:
  STRUCT
    fReal : REAL;
    bBool : ARRAY [0..2] OF BOOL;
    stInner : ST_INNTERSTRUCT;
  END_STRUCT
END_TYPE
```
Sample 1: Write operation with FB_XmlSrvWrite

In the first step the structure ST_MyStruct is to be written into an XML file. The mode is set to XMLSRV_ADMISSING, so that the XML file is generated automatically and the structure is created within it. Folders are not created automatically! For larger structures this approach is also recommend if the file is only to be read later. In this way the XML file does not have to be created manually, and errors are avoided.

(* Sample1 creates an XML-file under the path C:\Test.xml and writes value1 to it.
FUNCTIONBLOCK: FB_XmlSrvWrite *)

PROGRAM Sample1
VAR
value1 : ST_MyStruct;
fbXmlSrvWrite : FB_XmlSrvWrite;
bExecute : BOOL;
sFilePath : T_MaxString := 'C:\Test.xml'; (* CE: '\Hard Disk\Test.xml' *)
sXPath : T_MaxString := '/dataentry/MAIN.value1';
END_VAR

fbXmlSrvWrite(
  nMode := XMLSRV_ADMISSING,
pSymAddr := ADR(value1),
cbSymSize := SIZEOF(value1),
sFilePath := sFilePath,
sXPath := sXPath,
bExecute := bExecute
);
bExecute := TRUE;

Sample 2: Write operation with FB_XmlSrvWriteByName

Sample 2 leads to the same result as sample 1, but the block FB_XmlSrvWriteByName is used. Sample 1 offers higher performance.

(* Sample2 creates an XML-file under the path C:\Test.xml and writes value1 to it.
FUNCTIONBLOCK: FB_XmlSrvWriteByName *)

PROGRAM Sample2
VAR
value1 : ST_MyStruct;
fbXmlSrvWrite : FB_XmlSrvWriteByName;
bExecute : BOOL;
sSymName : T_MaxString := 'Sample2.value1';
sFilePath : T_MaxString := 'C:\Test.xml'; (* CE: '\Hard Disk\Test.xml' *)
sXPath : T_MaxString := '/dataentry/MAIN.value1';
END_VAR

fbXmlSrvWrite(
  nMode := XMLSRV_ADMISSING,
sSymName := sSymName,
sFilePath := sFilePath,
sXPath := sXPath,
bExecute := bExecute
);
bExecute := TRUE;

XML file

In both examples the XML file 'Test.XML' is created under C:\. In order to ensure that the XML file has the same content in both variants, the same sXPath is used, i.e. '/dataentry/MAIN.value', although value1 is not stored in Main, but directly in the respective programs. sSymName (Sample 2) specifies the location of the variables in TwinCAT: 'Sample2.value1'!

<?xml version="1.0" encoding="UTF-8"?>
<dataentry>
  <MAIN.value1>
    <fReal>0</fReal>
    <bBool index="0">false</bBool>
    <bBool index="1">false</bBool>
  </MAIN.value1>
</dataentry>
Sample 3: Read operation with FB_XmlSrvRead

The following section describes the process of reading the structure of the XML file created in sample 1 and/or sample 2. Sample 3 uses the block FB_XmlSrvRead for this purpose.

(* Sample3 reads an XML-file (C:\Test.xml)  
FUNCTIONBLOCK: FB_XmlSrvRead *)

PROGRAM Sample3

VAR
    value1 : ST_MyStruct;
    fbXmlSrvRead : FB_XmlSrvRead;
    bExecute : BOOL;
    sFilePath := 'C:\Test.xml'; (* CE: '\Hard Disk\Test.xml' *)
    sXPath := '/dataentry/MAIN.value1';
END_VAR

fbXmlSrvRead(
    pSymAddr := ADR(value1),
    cbSymSize := SIZEOF(value1),
    sFilePath := sFilePath,
    sXPath := sXPath,
    bExecute := bExecute
);

bExecute := TRUE;

Sample 4: Read operation with FB_XmlSrvReadByName

Sample 4 shows the read operation using the block FB_XmlSrvReadByName.

(* Sample4 reads an XML-file (C:\Test.xml)  
FUNCTIONBLOCK: FB_XmlSrvReadByName *)

PROGRAM Sample4

VAR
    value1 : ST_MyStruct;
    fbXmlSrvRead : FB_XmlSrvReadByName;
    bExecute : BOOL;
    sSymName := 'Sample4.value1';
    sFilePath := 'C:\Test.xml'; (* CE: '\Hard Disk\Test.xml' *)
    sXPath := '/dataentry/MAIN.value1';
END_VAR

fbXmlSrvRead(
    sSymName := sSymName,
    sFilePath := sFilePath,
    sXPath := sXPath,
    bExecute := bExecute
);

bExecute := TRUE;

Like for sample 2, note that sSymName and sXPath differ: sXPath indicates the path within the XML file. This was specified in sample 1 and/or sample 2. sSymName indicates the symbol name of the TwinCAT variable and is therefore 'Sample4.value1'.

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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>
5.3 Further Samples

The following samples show different types of application of the TC3 XML Server. The PLC-projekt which contains the samples can be downloaded here: https://infosys.beckhoff.com/content/1033/TF6421_Tc3_XML_Server/Resources/zip/1635013643.zip

Sample 5 shows an initialization once at program startup. Sample 6 shows cyclic and event-driven printing procedures. Both samples again use the structure ST_MYSTRUCT, which in turn includes ST_INNTERSTRUCT. Both Structures are shown in the following:

**Structures**

```
TYPE ST_MYSTRUCT:
  STRUCT
    fReal : REAL;
    bBool : ARRAY [0..2] OF BOOL;
    stInner : ST_INNTERSTRUCT;
  END_STRUCT
END_TYPE

TYPE ST_INNTERSTRUCT:
  STRUCT
    nInteger : INT;
    sString : STRING;
  END_STRUCT
END_TYPE
```

**Sample 5: One-time initialization at program startup**

Sample 5 shows the one-time initialization of value1 on program startup. The function block FB_XmlSrvRead is used.

```
(* Sample5 reads and initializes valuel when the PLC is started FUNCTIONBLOCK: FB_XmlSrvRead *)
PROGRAM Sample5
VAR
  value1 : ST_MyStruct;
  fbXmlSrvRead : FB_XmlSrvRead;
  bExecute : BOOL;
  sFilePath : T_MaxString := 'C:\Test.xml'; (* CE: 'Hard Disk\Test.xml' *)
  sXPath : T_MaxString := '/dataentry/MAIN.value1';
  nState : INT := 0;
END_VAR
CASE nState OF
  0: (* initialize *)
    fbXmlSrvRead(
      pSymAddr := ADR(value1),
      cbSymSize := SIZEOF(value1),
      sFilePath := sFilePath,
      sXPath := sXPath,
      bExecute := bExecute
    );
    fbXmlSrvRead(bExecute:= TRUE);
    nState:= 1;
  1: (* wait for read operation *)
    fbXmlSrvRead(bExecute:= FALSE);
    IF NOT fbXmlSrvRead.bBusy AND NOT fbXmlSrvRead.bError THEN
      nState:= 2;
    ELSIF fbXmlSrvRead.bError THEN
      nState:= 100;
    END_IF
  2: (* operations *)
  100: (* errorState *)
END_CASE
```
Sample 6: Cyclic and event-driven writing

The following example generates a new XML file every 20 seconds and writes the familiar structure into it. As usual, the file name is generated based on the current Windows date and time and a string. The write process can also be started by pressing switch (or by setting the switch variable bButton). If the write process is triggered several times within one second, the current file is overwritten.

/* Sample6: Every 20s value1 will be written into a new XML-File named after the current date and time. Furthermore you can activate the printing procedure by pressing a button (or setting the corresponding variable *)

PROGRAM Sample6
VAR
  value1 : ST_MyStruct;
  fbXmlSrvWrite : FB_XmlSrvWrite;
  sFileFolder : T_MaxString := 'C:\'; (* CE: \Hard Disk\ *)
  sFileName : T_MaxString := '_test.xml';
  sFilePathWrite : T_MaxString
  (*sFilePathWrite = sFileFolder + time + sFileName*)
  sXPathWrite : T_MaxString := '/dataentry/MAIN.value1';
  ntGetTime : NT_GetTime;
  stMyTimestruct : TIMESTRUCT;
  iState : INT := 1;
  bTwentySec : BOOL:= FALSE;
  bButton : BOOL:= FALSE;
  bTwentySecOver : BOOL;
  triggerWrite : R_TRIG;
  triggerButton : R_TRIG;
END_VAR

triggerButton(CLK:= bButton);

CASE iState OF
  0: (* idle state *)
  
  1: (* initialize *)
    fbXmlSrvWrite(nMode:=XMLSRV_ADDMISSING, pSymAddr:= ADR(value1),
    cbSymSize:= SIZEOF(value1));
    ntGetTime(START:= TRUE, TIMESTR=>stMyTimestruct); (* get Windows time *)
    IF NOT ntGetTime.BUSY AND NOT ntGetTime.ERR THEN
      iState:= 2;
    ELSIF ntGetTime.ERR THEN
      iState:= 100;
    END_IF
    (* change some values - replace with production-process *)
    value1.stInner.nInteger:= value1.stInner.nInteger + 1;
    IF value1.stInner.nInteger = 32767 THEN
      value1.stInner.nInteger:= 0;
    END_IF(* get Windows time *)
    ntGetTime(START:= FALSE);
    IF NOT ntGetTime.BUSY AND NOT ntGetTime.ERR THEN
      ntGetTime(START:= TRUE, TIMESTR=>stMyTimestruct);
      ELSE
        iState:= 100;
      END_IF
      (* check if 20s have passed*)
      IF stMyTimestruct.wSecond = 0 OR stMyTimestruct.wSecond = 20
        OR stMyTimeStruct.wSecond = 40 THEN
        bTwentySecOver:= TRUE;
      ELSE
        bTwentySecOver:= FALSE;
      END_IF
      (* if 20s have passed => trigger writing-process *)
      triggerWrite(CLK:=bTwentySecOver);
      IF (triggerWrite.Q OR triggerButton.Q) AND NOT fbXmlSrvWrite.bBusy AND NOT fbXmlSrvWrite.bError THEN
        (* create filename *)
        sFilePathWrite:= CONCAT(sFileFolder, SYSTEMTIME_TO_STRING(stMyTimestruct)); (* set folder + time *)
        sFilePathWrite:= DELETE(STR:= sFilePathWrite, LEN:= 4 , POS:= LEN(STR:=sFilePathWrite)-3); (*
delete milliseconds *
    sFilePathWrite:= REPLACE(STR1:= sFilePathWrite , STR2:= '.' , L:= 1,
    P:= LEN(STR:=sFilePathWrite)-2); (* replace colon with point *)
    sFilePathWrite:= REPLACE(STR1:= sFilePathWrite , STR2:= '.' , L:= 1,
    P:= LEN(STR:=sFilePathWrite)-5); (* replace colon with point *)
    sFilePathWrite:= CONCAT(sFilePathWrite, sFileName); (* add filename (default: test) *)
    (* change value 1*)
    value1.stInner.sString := sFilePathWrite;
    value1.stInner.nInteger := stMyTimestruct.wSecond;

    (* write *)
    fbXmlSrvWrite(sFilePath:=sFilePathWrite, sXPath:=sXPathWrite, bExecute:= TRUE);

ELSIF fbXmlSrvWrite.bError THEN
    iState:= 100;
END_IF

    (* reset fbXmlSrvWrite *)
    IF fbXmlSrvWrite.bBusy AND NOT tGetTime.ERR THEN
        fbXmlSrvWrite(bExecute:= FALSE);
    ELSIF ntGetTime.ERR THEN
        iState:= 100;
    END_IF

100: (* error state*)
;
END_CASE

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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>

5.4 Production example

Sample 7 (production example)

This section describes an example for a production order. Production data such as component length, component width, etc. are read in a structure for initialization (XML read), the production takes place with a corresponding quantity, and the production order is completed with an entry in the XML file (write). To start the program, the XML file first has to be saved in the location that matches the file path, and in the PLC program the variable bStart has to be set to TRUE.

The PLC project containing the examples can be downloaded from here: https://infosys.beckhoff.com/content/1033/TF6421_Tc3_XML_Server/Resources/zip/1635013643.zip

Variable declaration

PROGRAM Sample7
VAR
    fbXmlSrvReadByName   : FB_XmlSrvReadByName;
    fbXmlSrvWriteByName  : FB_XmlSrvWriteByName;
    value                : ST_MyProductionStruct;
    state                : INT := 0;
    R_Edge               : R_TRIG;
    bStart               : BOOL;
    bError               : BOOL;
    nErrId               : UDINT;
END_VAR

Structure ST_MyProductionStruct

TYPE ST_MyProductionStruct :
    STRUCT
        rLength : REAL;
        rWidth  : REAL;
        rHeight : REAL;
        iQuantity : INT;
        iCounter : INT;
    END_STRUCT
PLC program

(* The production data is read, the production is carried out and, according to the
quantity and the production order, completed with an entry in the XML file.
To start the program the XML file needs to be stored in the corresponding folder (sFilePath) and the
variable bStart needs to be set TRUE in the PLC program. *)

R_Edge (CLK := bStart);
IF R_Edge.Q THEN
    state := 1;
END_IF

CASE state OF
    0: (* idle state *)
        
    1: (* init state *)
        fbXmlSrvReadBySource( sNetId := '',
                       sSymName := 'Sample7.value',
                       sFilePath := 'C:\Production1.xml',
                       sXPath := '/dataentry/MAIN.value',
                       bExecute := TRUE,
                       tTimeout := t#10s,
                       bError => bError,
                       nErrId => nErrId);
    state := 2;

    2:
        fbXmlSrvReadBySource(bExecute := FALSE);
        IF NOT fbXmlSrvReadBySource.bBusy AND NOT fbXmlSrvReadBySource.bError THEN
            state := 3;
        ELSIF fbXmlSrvReadBySource.bError THEN
            state := 100;
        END_IF

    3: (* working state *)
        IF value.bReady = TRUE THEN
            value.stInfo := 'The order was already processed!';
            (* replace your production XML file! *)
            state := 4;
        RETURN;
    END_IF

    (* call production program with
    new length, width and height here *)
        value.iCounter := value.iCounter + 1;
        IF value.iCounter = value.iQuantity THEN
            value.bReady := TRUE;
            state := 4;
    END_IF

    4: (* documentation state *)
        fbXmlSrvWriteBySource( sNetId := '',
                       nMode := XMLSRV_SKIPMISSING,
                       sSymName := 'Sample7.value',
                       sFilePath := 'C:\Production1.xml',
                       sXPath := '/dataentry/MAIN.value',
                       bExecute := TRUE,
                       tTimeout := t#10s,
                       bError => bError,
                       nErrId => nErrId);
    state := 5;

    5:
        fbXmlSrvWriteBySource(bExecute := FALSE);
        IF NOT fbXmlSrvWriteBySource.bBusy AND NOT fbXmlSrvWriteBySource.bError THEN
            state := 0;
        ELSIF fbXmlSrvWriteBySource.bError THEN
            state := 100;
        END_IF
XML file

```xml
<dataentry>
  <MAIN.value>
    <rLength>65.85</rLength>
    <rWidth>30</rWidth>
    <rHeight>2.5</rHeight>
    <iQuantity>500</iQuantity>
    <iCounter>0</iCounter>
    <bReady>false</bReady>
    <stInfo></stInfo>
  </MAIN.value>
</dataentry>
```

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 v3.1 Build 4011</td>
<td>PC or CX (x86, x64, ARM)</td>
<td>Tc2_XmlDataSrv</td>
</tr>
</tbody>
</table>
# Appendix

## 6.1 Overview of TC3 XML Server error codes

<table>
<thead>
<tr>
<th>Offset + error code</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000000000 + TwinCAT system error codes</td>
<td>0x00000000-0x00007800</td>
<td>TwinCAT system error (including ADS error codes)</td>
</tr>
<tr>
<td>0x00008000 + internal TC3 XML Server error [32]</td>
<td>0x00008000-0x000080FF</td>
<td>Internal TC3 XML Server error codes</td>
</tr>
</tbody>
</table>

## 6.2 Internal error codes of the TwinCAT 3 XML Server

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Symbolic name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00008000</td>
<td>Internal error</td>
<td>XMLSRVERR_INTERNAL</td>
</tr>
<tr>
<td>0x00008001</td>
<td>Handle not found.</td>
<td>XMLSRVERR_NOTFOUND</td>
</tr>
<tr>
<td>0x00008002</td>
<td>Error during parsing of the XML file</td>
<td>XMLSRVERR_PARSERERROR</td>
</tr>
<tr>
<td>0x00008003</td>
<td>Incompatible data type.</td>
<td>XMLSRVERR_INCOMPATIBLE</td>
</tr>
<tr>
<td>0x00008004</td>
<td>Error during memory allocation</td>
<td>XMLSRVERR_NOMEMORY</td>
</tr>
<tr>
<td>0x00008005</td>
<td>Error during adding of an XML node</td>
<td>XMLSRVERR_ADDNODE</td>
</tr>
<tr>
<td>0x00008006</td>
<td>Invalid sXPath</td>
<td>XMLSRVERR_INVALIDXPATH</td>
</tr>
<tr>
<td>0x00008007</td>
<td>Invalid string in TC</td>
<td>XMLSRVERR_INVALIDSTRING</td>
</tr>
<tr>
<td>0x00008000</td>
<td>Invalid handle of the client</td>
<td>XMLSRVERR_INVALIDCLIENTHANDLE</td>
</tr>
<tr>
<td>0x00089000</td>
<td>Use of a wrong TcXmlDataSrv.exe (for TC2 than TC3)</td>
<td>XMLSRVERR_INVALIDTWINCATVERSION</td>
</tr>
</tbody>
</table>

## 6.3 FAQ - Frequently asked questions and their answers

In this area we answer frequently asked questions to help you to work with TF6421 XML Server.

If you have any further questions please contact our support +49(0)5246/963-157.

What kind of information is saved to the XML file? [32]

Is it possible to write several variables into an XML file simultaneously? [32]

Is there a way to access an XML file in the network? [33]

Does the XML-Server have a feature to automatically create files? [33]

What happens if the XML Server is not able to convert a variable, because the types do not match (e.g. an INT has the value “hello” in an XML file)? [33]

What kind of information is saved to the XML file?

The only information saved to the XML-file is the name of the variable and the value. There is no information about the data type saved. You can find more information on that in the section “Getting Started [23].”

Is it possible to write several variables into an XML file simultaneously?

Yes, this is possible. Therefore you have to wrap up your variables in a structure. You then write a variable of this type. This is shown in the samples [23].
? **Is there a way to access an XML file in the network?**
! No, this is not possible. sPath may only point at the local filesystem.

? **Does the XML-Server have a feature to automatically create files?**
! Yes, it does. To use this feature you have to set nMode:=1 (XMLSRV_ADDMISSING). The XML Server will create files and parts in the file automatically. Folders will not be created!

? **What happens if the XML Server is not able to convert a variable, because the types do not match (e.g. an INT has the value “hallo” in an XML file)?**
! In this case this variable will just be skipped. It does not lead to an error.