Manual

TC3 Modbus RTU

TwinCAT 3

Version	1.0
Date	2014-06-17
Order No.	TF6255

BECKHOFF

Table of contents

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with the applicable national standards.
It is essential that the following notes and explanations are followed when installing and commissioning these components.

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.
For that reason the documentation is not in every case checked for consistency with performance data, standards or other characteristics.
In the event that it contains technical or editorial errors, we retain the right to make alterations at any time and without warning.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

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

Patent Pending

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

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

EtherCAT®

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

Copyright

1.2 Safety instructions

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

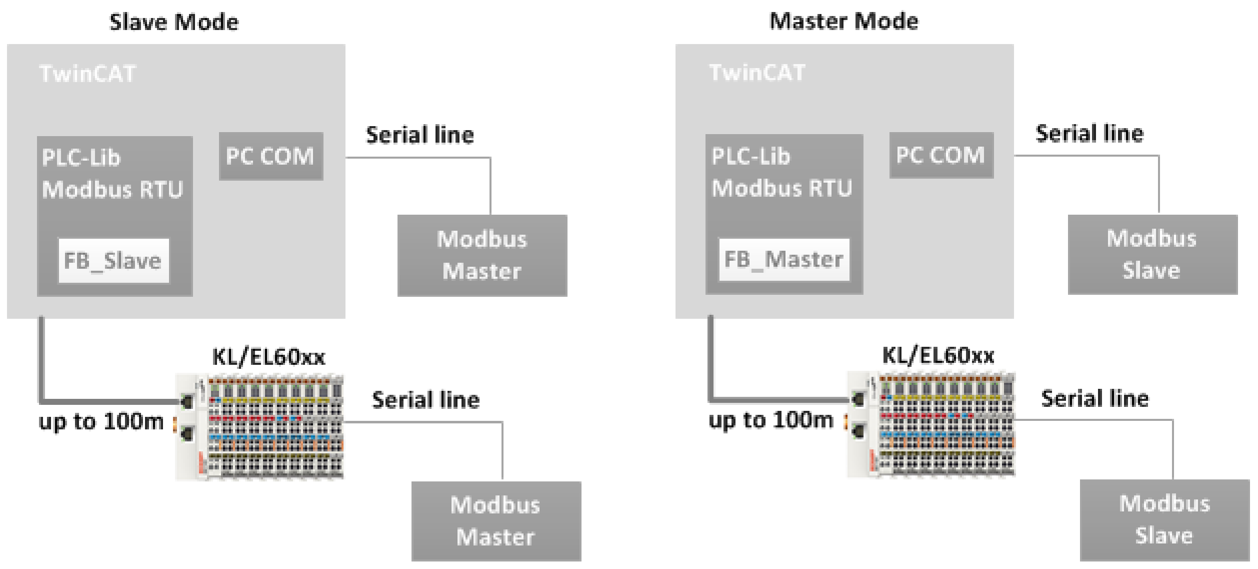
Description of symbols

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

⚠ DANGER	**Serious risk of injury!** Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.
⚠ WARNING	**Risk of injury!** Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.
⚠ CAUTION	**Personal injuries!** Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.
❗ Attention	**Damage to the environment or devices** Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.
ℹ Note	**Tip or pointer** This symbol indicates information that contributes to better understanding.

2 Overview

The TwinCAT 3 Modbus RTU offers function blocks for serial communication with Modbus end devices.



Modbus RTU devices can be connected to a TwinCAT controller via a serial interface. The TwinCAT PLC uses the slave function blocks of the Modbus RTU library to communicate with the Modbus master (Slave Mode). Alternatively, Modbus master functions are available, which can address one or several Modbus slaves (Master Mode).

Supported devices

- Serial COM-Port of a PC or CX
- Serial COM-Port of a Beckhoff BX Controller
- Serial KL-Terminals KL60xx
- Serial EtherCAT-Terminal EL60xx

Further documentation

Technical details and specification about Modbus can be found under: http://www.modbus.org

3 Installation

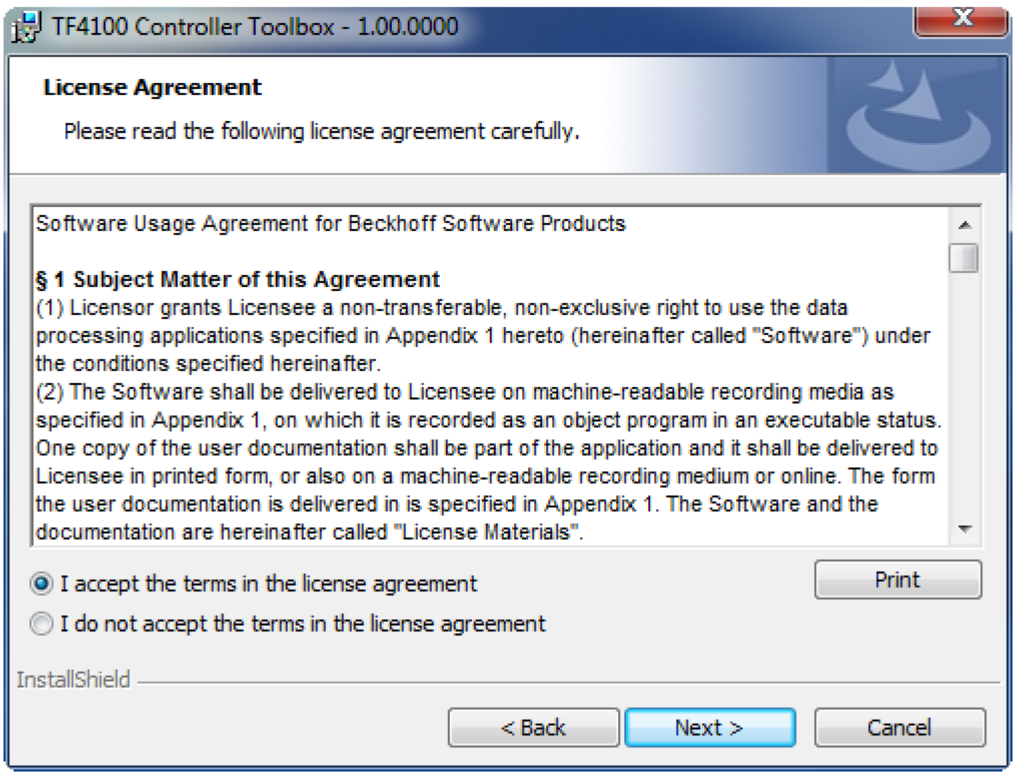
3.1 System Requirements

Technical Data	TF6255 TC3 Modbus-RTU
Target System	Windows XP / 7 PC or CX (x86)
Min. TwinCAT-Version	3.0.3101
Min. TwinCAT-Level	TC1200 TC3 PLC

3.2 Installation

Description of the installation procedure of a TwinCAT 3 Function for Windows-based operating Systems.

1. Double-click the downloaded setup file "TFxxxx".
   Please note: Under Windows 32-bit/64-bit, please start the installation with "Run as Administrator" by right-clicking the setup file and selecting the corresponding option in the context menu.
2. Click on "Next" and accept the license Agreement.

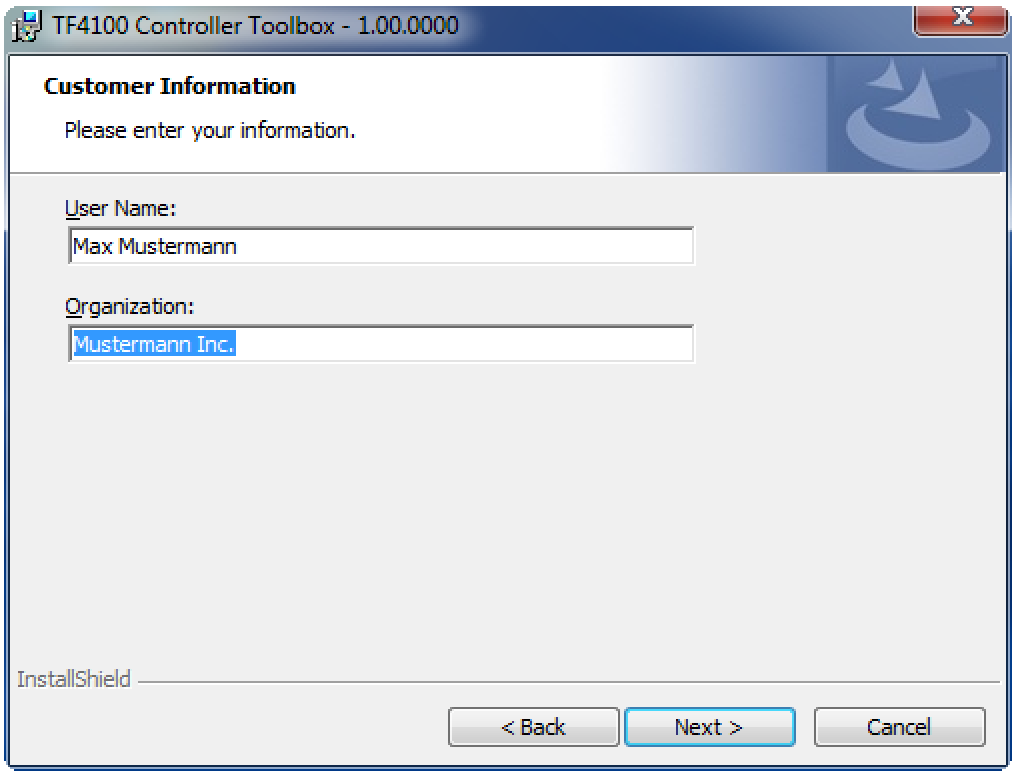

TF4100 Controller Toolbox - 1.00.0000
License Agreement
Please read the following license agreement carefully.

Software Usage Agreement for Beckhoff Software Products

§ 1 Subject Matter of this Agreement
(1) Licensor grants Licensee a non-transferable, non-exclusive right to use the data processing applications specified in Appendix 1 hereto (hereinafter called "Software") under the conditions specified hereinafter.
(2) The Software shall be delivered to Licensee on machine-readable recording media as specified in Appendix 1, on which it is recorded as an object program in an executable status. One copy of the user documentation shall be part of the application and it shall be delivered to Licensee in printed form, or also on a machine-readable recording medium or online. The form the user documentation is delivered in is specified in Appendix 1. The Software and the documentation are hereinafter called "License Materials".

[✓ I accept the terms in the license agreement]
I do not accept the terms in the license agreement

3. Enter your user information in the specified area.



TF4100 Controller Toolbox - 1.00.0000

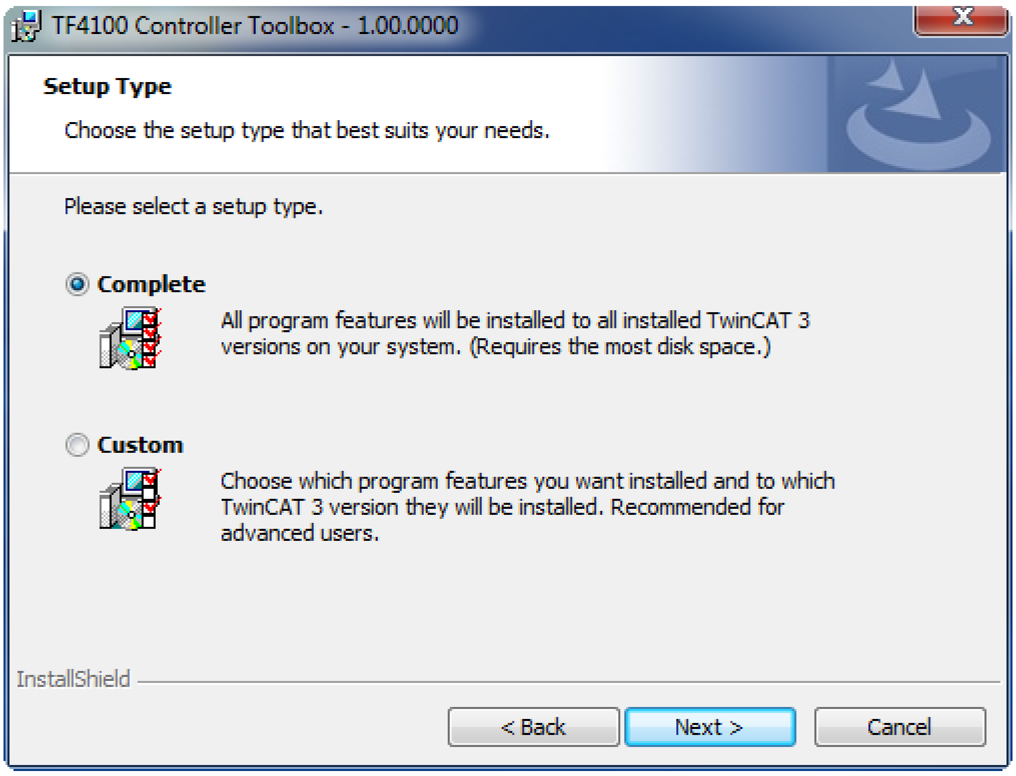
Customer Information

Please enter your information.

User Name: Max Mustermann

Organization: Mustermann Inc.

4. To install the full product, including all sub-components, please choose **"Complete"** as the Setup Type.Alternatively you can also install each component seperately by choosing **"Custom"**.



TF4100 Controller Toolbox - 1.00.0000

Setup Type

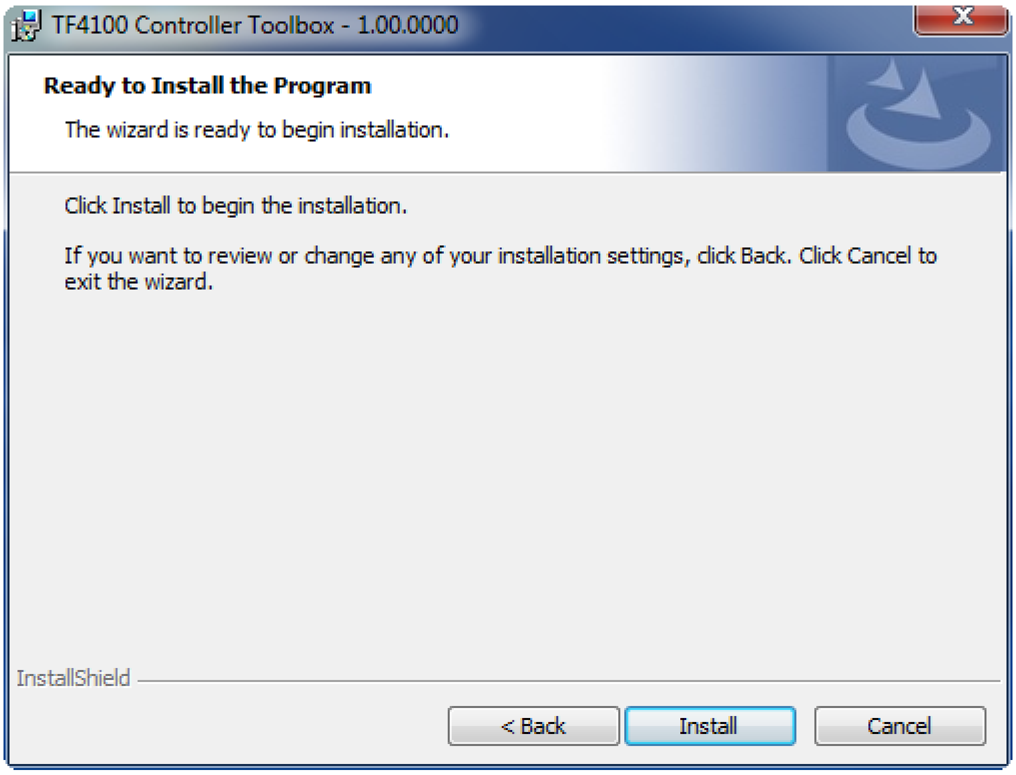
Choose the setup type that best suits your needs.

Please select a setup type.

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

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

5. Click on **"Install"**after pressing the **"Next"** to start the Installation.


TF4100 Controller Toolbox - 1.00.0000

Ready to Install the Program

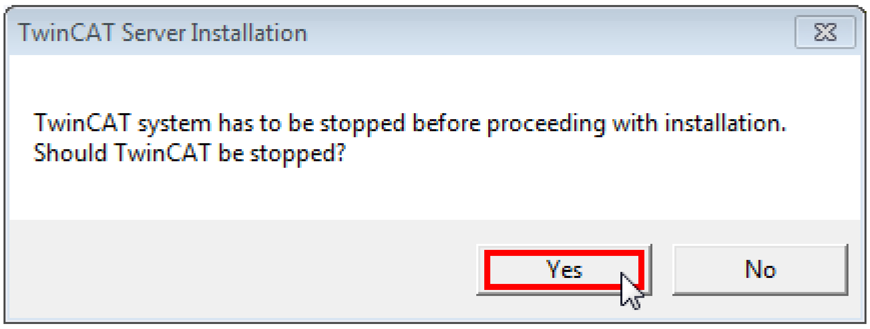
The wizard is ready to begin installation.

Click Install to begin the installation.

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

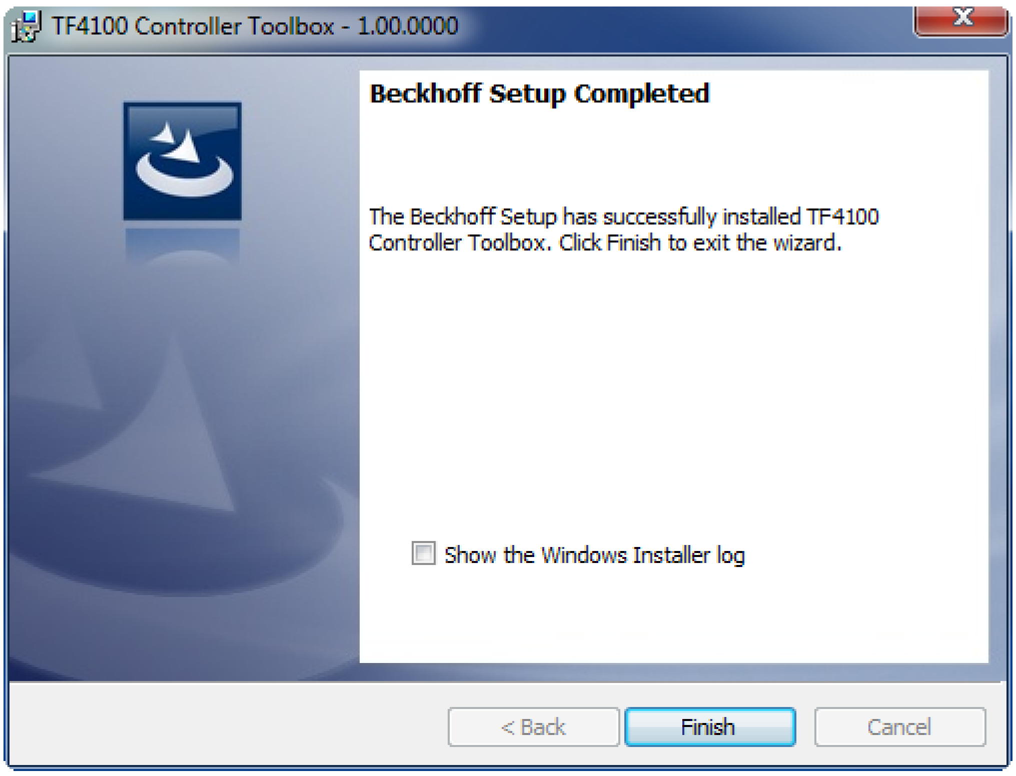
The TwinCAT system has to be stopped before proceeding with installation

6. Confirm the Dialog with **"Yes"**.


TwinCAT Server Installation

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

7. Select **"Finish"** to end the installation process.



TF4100 Controller Toolbox - 1.00.0000

Beckhoff Setup Completed

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

☐ Show the Windows Installer log

⇨ The installation is complete now.

After a successful installation the TC 3Function needs to be licensed [▹ 10].

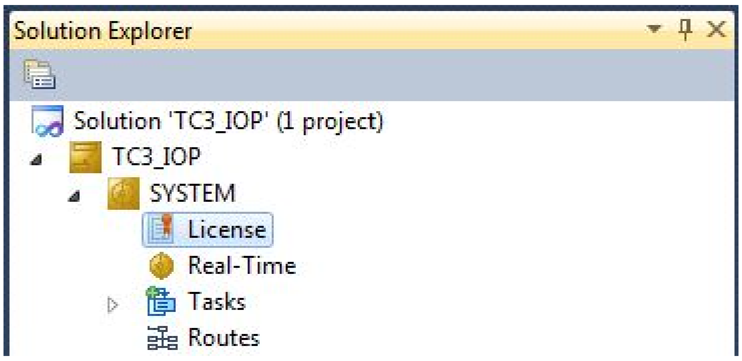
3.3 Licensing

The TwinCAT 3 functions are available both as a full and as a 7-Day trial version. Both license types can be activated via TwinCAT XAE.For more information about TwinCAT 3 licensing, please consult the TwinCAT 3 Help System.The following document describes both licensing scenarios for a TwinCAT 3 function on TwinCAT 3 and is divided into the following sections:

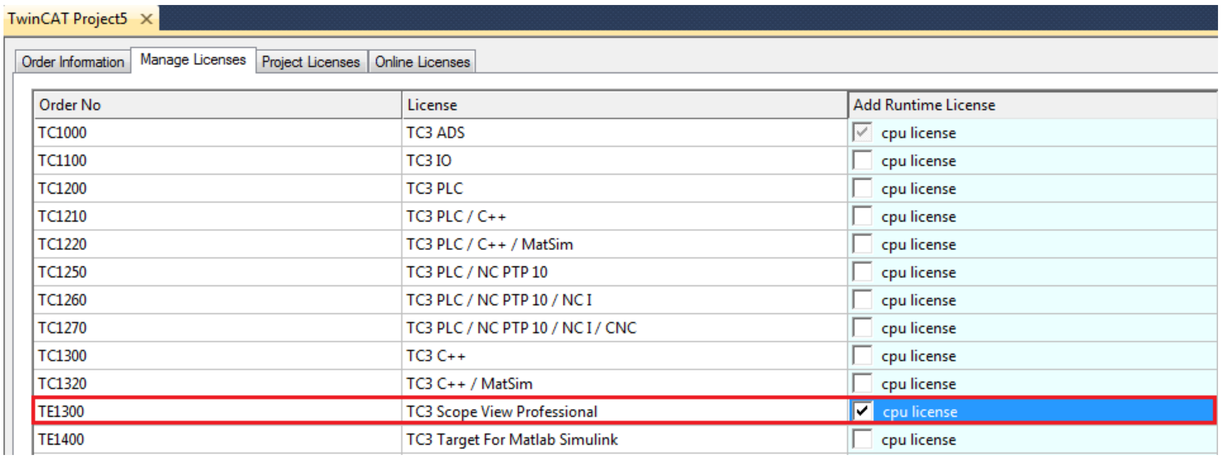
- Licensing a 7-Day trial version [▹ 10]
- Licensing a full version [▹ 11]

Licensing a 7-Day trial version

1. Start TwinCAT XAE
2. Open an existing TwinCAT 3 project or create a new project
3. In “Solution Explorer”, please navigate to the entry **“System\License”**

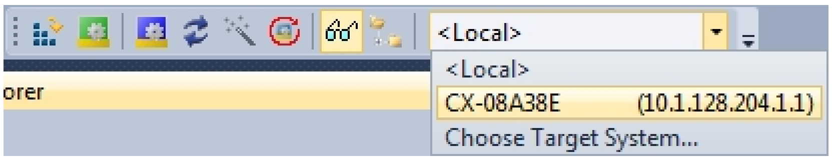


4. Open the tab **"Manage Licenses"** and add a **"Runtime License"** for your product (in this screenshot "TE1300: TC3 Scope View Professional")

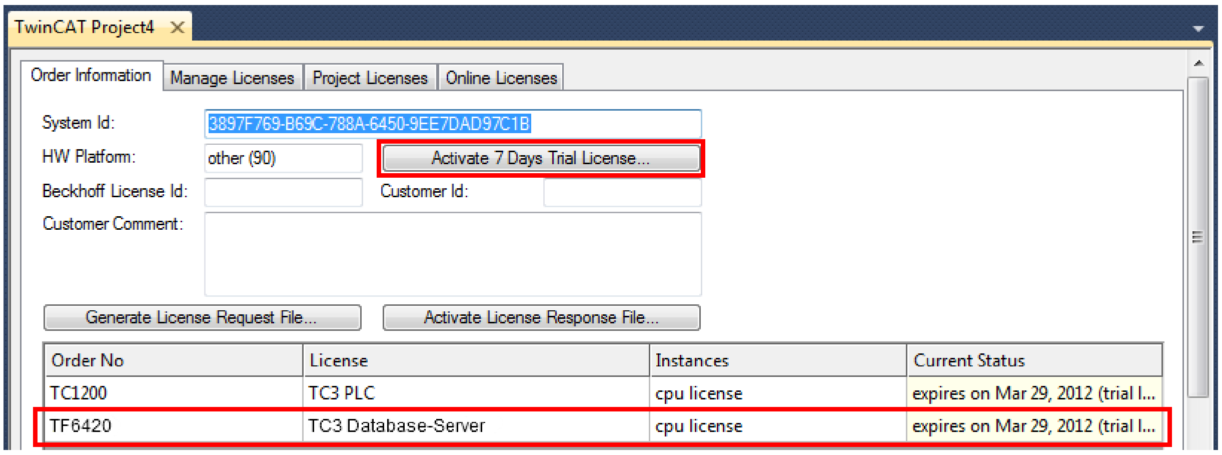


Order No	License	Add Runtime License
TC1000	TC3 ADS	✓ cpu license
TC1100	TC3 IO	cpu license
TC1200	TC3 PLC	cpu license
TC1210	TC3 PLC / C++	cpu license
TC1220	TC3 PLC / C++ / MatSim	cpu license
TC1250	TC3 PLC / NC PTP 10	cpu license
TC1260	TC3 PLC / NC PTP 10 / NC I	cpu license
TC1270	TC3 PLC / NC PTP 10 / NC I / CNC	cpu license
TC1300	TC3 C++	cpu license
TC1320	TC3 C++ / MatSim	cpu license
TE1300	TC3 Scope View Professional	✓ cpu license
TE1400	TC3 Target For Matlab Simulink	cpu license

5. **Optional**: If you would like to add a license for a remote device, you first need to connect to the remote device via TwinCAT XAE toolbar



6. Switch to the tab **"Order Information"** and click the button **"Activate 7 Days Trial License..."**to activate a test version


TwinCAT Project4
Order Information
Manage Licenses
Project Licenses
Online Licenses
Activate 7 Days Trial License...
Generate License Request File...
Activate License Response File...

System Id: 3897F769-B69C-788A-6450-9EE7DAD97C1B
HW Platform: other (90)
Beckhoff License Id:
Customer Id:
Customer Comment:

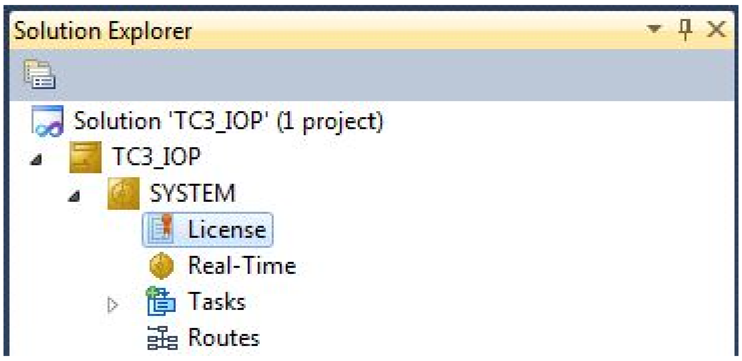
Order No	License	Instances	Current Status
TC1200	TC3 PLC	cpu license	expires on Mar 29, 2012 (trial l...
TF6420	TC3 Database-Server	cpu license	expires on Mar 29, 2012 (trial l...

7. Please restart TwinCAT 3 afterwards.

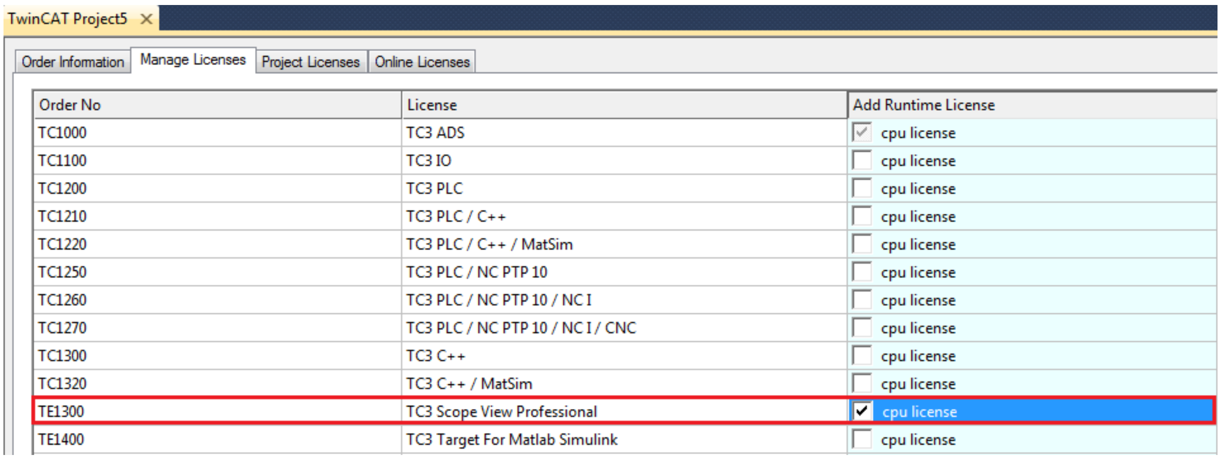
Licensing a full version

8. Start TwinCAT XAE
9. Open an existing TwinCAT 3 project or create a new project

10. In "Solution Explorer", please navigate to the entry **"SYSTEM\License"**

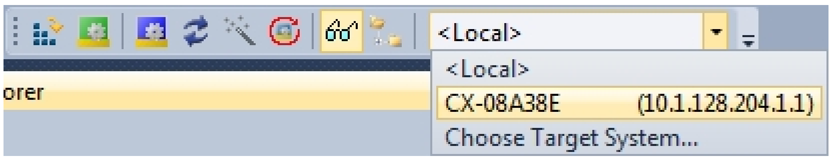


11. Open the tab **"Manage Licenses"** and add a **"Runtime License"** for your product (in this screenshot " TE1300: TC3 Scope View Professional”).



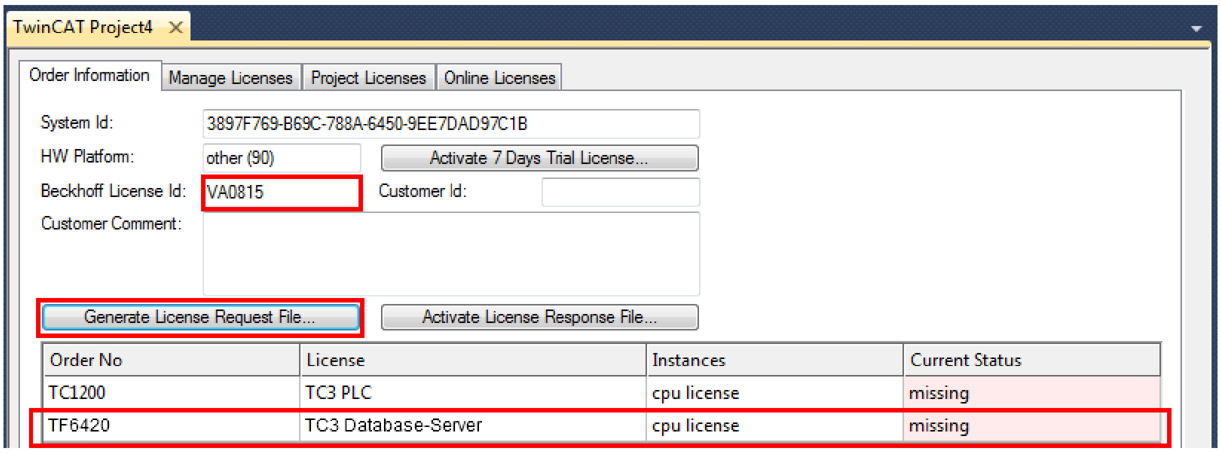
Order No	License	Add Runtime License
TC1000	TC3 ADS	✓ cpu license
TC1100	TC3 IO	cpu license
TC1200	TC3 PLC	cpu license
TC1210	TC3 PLC / C++	cpu license
TC1220	TC3 PLC / C++ / MatSim	cpu license
TC1250	TC3 PLC / NC PTP 10	cpu license
TC1260	TC3 PLC / NC PTP 10 / NC I	cpu license
TC1270	TC3 PLC / NC PTP 10 / NC I / CNC	cpu license
TC1300	TC3 C++	cpu license
TC1320	TC3 C++ / MatSim	cpu license
TE1300	TC3 Scope View Professional	✓ cpu license
TE1400	TC3 Target For Matlab Simulink	cpu license

12. **Optional:**If you would like to add a license for a remote device, you first need to connect to the remote device via TwinCAT XAE toolbar



13. Navigate to the **"Order Information"** tab
The fields "System-ID" and "HW Platform" cannot be changed and just describe the platform for the licensing process in general a TwinCAT 3 license is always bound to these two identifiers:
the "System-ID" uniquely identifies your system.
The "HW Platform" is an indicator for the performance of the device.

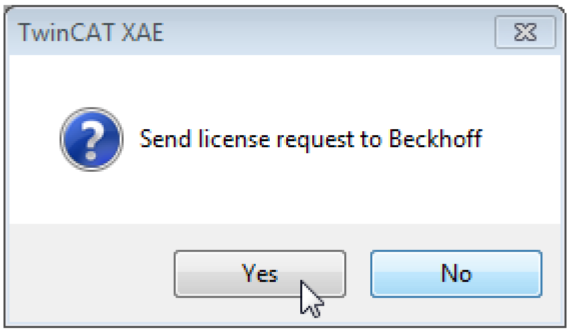
14. Optionally, you may also enter an own order number and description for your convenience


TwinCAT Project4
Order Information
Manage Licenses
Project Licenses
Online Licenses
Activate 7 Days Trial License...
Generate License Request File...
Activate License Response File...

System Id: 3897F769-B69C-788A-6450-9EE7DAD97C1B
HW Platform: other (90)
Beckhoff License Id: VA0815
Customer Id:
Customer Comment:

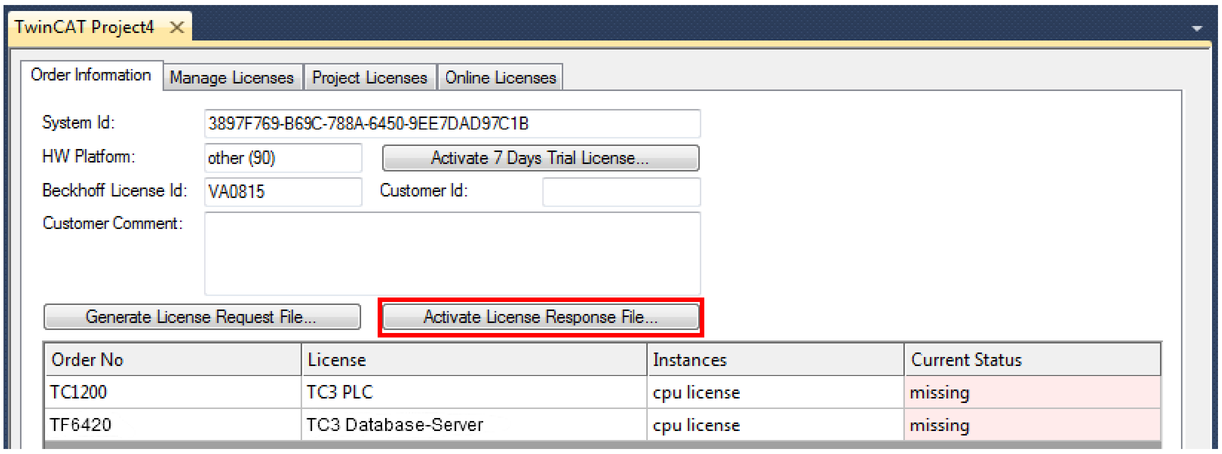
Order No	License	Instances	Current Status
TC1200	TC3 PLC	cpu license	missing
TF6420	TC3 Database-Server	cpu license	missing

15. enter the "Beckhoff License ID" and click on **"Generate License Request File..."**. If you are not aware of your **"Beckhoff License ID"** please contact your local sales representative.
16. After the license request file has been saved, the system asks whether to send this file via E-Mail to the Beckhoff Activation Server


TwinCAT XAE

Send license request to Beckhoff

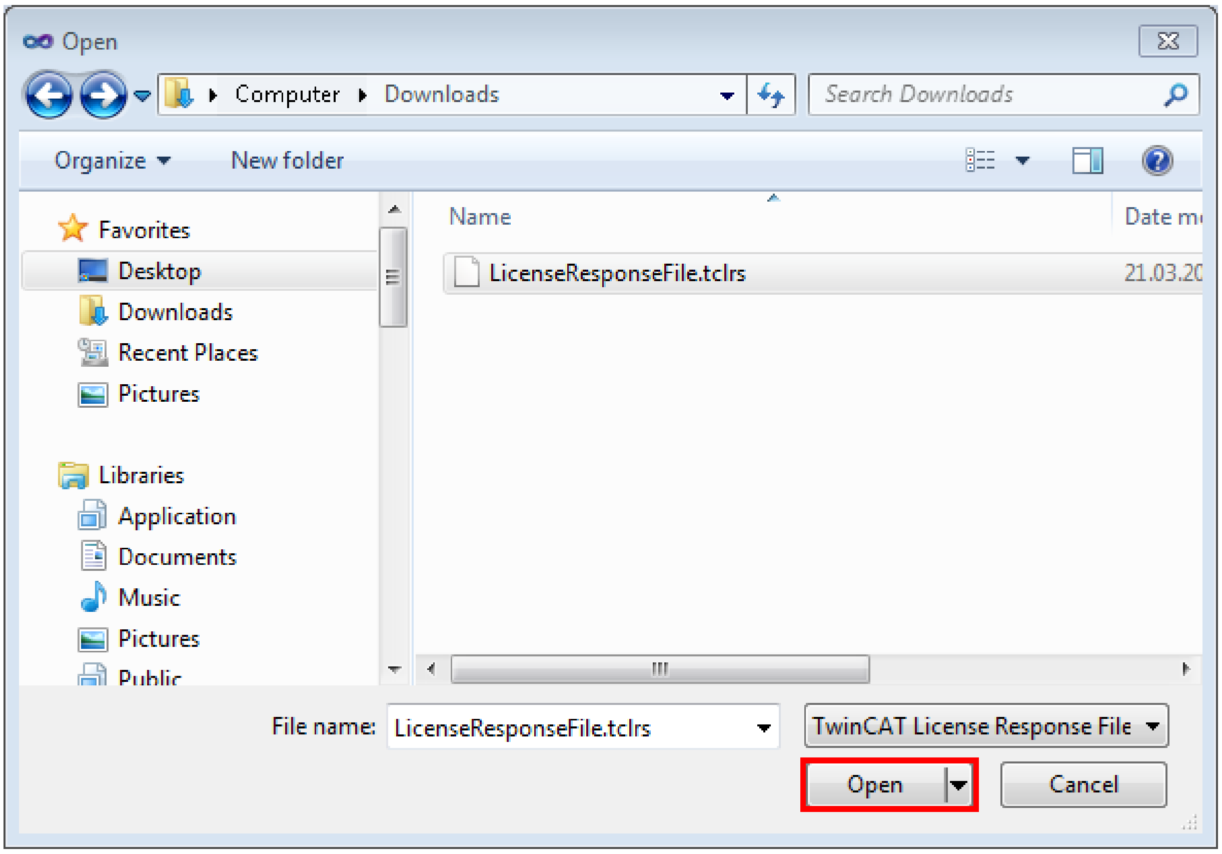
17. After clicking "Yes", the standard E-Mail client opens and creates a new E-Mail message to "tclicense@beckhoff.com" which contains the "License Request File"
18. Send this Activation Request to Beckhoff
    NOTE! The "License Response File" will be sent to the same E-Mail address used for sending out the "License Request File"
19. After receiving the activation file, please click on the button "Activate License Response File..."in the TwinCAT XAE license Interface.


TwinCAT Project4
Order Information
Manage Licenses
Project Licenses
Online Licenses
Activate 7 Days Trial License...
Generate License Request File...
Activate License Response File...

System Id: 3897F769-B69C-788A-6450-9EE7DAD97C1B
HW Platform: other (90)
Beckhoff License Id: VA0815
Customer Id:
Customer Comment:

Order No	License	Instances	Current Status
TC1200	TC3 PLC	cpu license	missing
TF6420	TC3 Database-Server	cpu license	missing

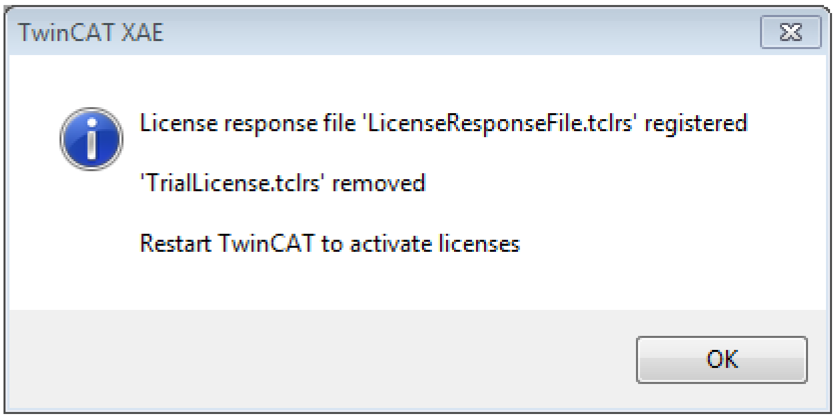
20. Select the received "Licnse response file" and click on "Open"



File name: LicenseResponseFile.tclrs

TwinCAT License Response File

21. The "License Response File" will be imported and all included licenses will be activated. If there have been any trial licenses, these will be removed accordingly.
22. Please restart TwinCAT to activate licenses..



License response file 'LicenseResponseFile.tclrs' registered

'TrialLicense.tclrs' removed

Restart TwinCAT to activate licenses

NOTE! The license file will be automatically copied to "..\TwinCAT\3.1\Target\License" on the local device.

4 Configuration

4.1 Terminal configuration

The Bus Terminals KL6001, KL6011, KL6021, KL6031 and KL6041 can be parameterized with the KS2000 configuration software. Alternatively, the system can be configured via PLC blocks included in the serial communication library ComLib.lib. If the serial communication library is not used in conjunction with the Modbus RTU library, the basic library *KL6Config.lib*, which is supplied with the Modbus RTU library, can be integrated. This library contains the following blocks from the serial communication library.

- KL6configuration
- KL6ReadRegisters
- KL6WriteRegisters
- ComReset

4.2 Modbus address arrays

Modbus defines access functions for different data arrays. These data arrays are declared as variables in a TwinCAT PLC program, e.g. as word arrays, and transferred to the Modbus slave function block as input parameters. Each array has a different Modbus start address, so that the arrays can be distinguished unambiguously. This offset has to be taken account of for addressing.

Inputs

The *Inputs* data array usually describes the physical input data with read-only access. They can be digital inputs (bit) or analog inputs (word). The PLC programmer can decide whether or not to grant the communication partner direct access to the physical inputs. It is also possible to define an input array for Modbus communication that is not identical with the physical inputs:

Definition of the Modbus input data as direct image of the physical inputs. Start and size of the data array can be specified freely. They are limited by the actual size of the input process image of the controller used.

```
VAR
Inputs AT%IW0 : ARRAY[0..255] OF WORD;
END_VAR
```

Definition of the Modbus input data as a separate Modbus data array independent of the physical inputs

```
VAR
Inputs : ARRAY[0..255] OF WORD;
END_VAR
```

Access to the *Input* array via a Modbus master is possible with the following Modbus functions:

```
2 : Read Input Status
4 : Read Input Registers
```

Addressing

The *Input* array is addressed with a 0 offset, i.e. address 0 as transferred in the telegram addresses the first element in the input data array.

Examples:

PLC variable	Access type	Address in the Modbus telegram	Address in the end device (device-dependent)
Inputs[0]	Word	16#0	30001
Inputs[1]	Word	16#1	30002
Inputs[0], Bit 0	Bit	16#0	10001
Inputs[1], Bit 14	Bit	16#1E	1001F

Outputs

The *Outputs* data array usually describes the physical output data with read and write access. *Outputs* can be digital outputs (coils) or analog outputs (output registers). Like for the *Inputs*, the array can be declared as a physical output variable or as a simple variable.

Definition of the Modbus output data as direct image of the physical outputs. Start and size of the data array can be specified freely. They are limited by the actual size of the output process image of the controller used.

```
VAR
Outputs AT%QW0 : ARRAY[0..255] OF WORD;
END_VAR
```

Definition of the Modbus output data as a separate Modbus data array independent of the physical outputs

```
VAR
Outputs : ARRAY[0..255] OF WORD;
END_VAR
```

Access to the *Output* array via a Modbus master is possible with the following Modbus functions:

```
1 : Read Coil Status
3 : Read Holding Registers
5 : Force Single Coil
6 : Preset Single Register
15 : Force Multiple Coils
16 : Preset Multiple Registers
```

Addressing

The *Output* array is addressed with a 16#800 offset, i.e. address 16#800 as transferred in the telegram addresses the first element in the output data array.

Examples:

PLC variable	Access type	Address in the Modbus telegram	Address in the end device (device-dependent)
Outputs[0]	Word	16#800	40801
Outputs[1]	Word	16#801	40802
Outputs[0], Bit 0	Bit	16#800	00801
Outputs[1], Bit 14	Bit	16#81E	0081F

Memory

The *Memory* data array describes a PLC variable array without physical I/O assignment.

Definition of the Modbus memory data as PLC flags. Start and size of the data array can be specified freely.

```
VAR
Memory AT%MW0 : ARRAY[0..255] OF WORD;
END_VAR
```

Definition of the Modbus memory data as variable without flag address

```
VAR
Memory : ARRAY[0..255] OF WORD;
END_VAR
```

Access to the *Memory* array via a Modbus master is possible with the following Modbus functions:

```
3 : Read Holding Registers
6 : Preset Single Register
16 : Preset Multiple Registers
```

Addressing

The *Memory* array is addressed with a 16#4000 offset, i.e. address 16#4000 as transferred in the telegram addresses the first word in the output data array.

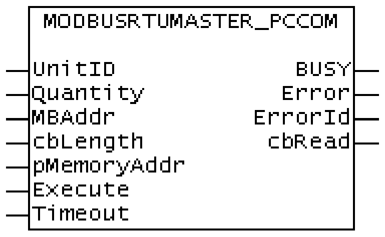
Examples:

PLC variable	Access type	Address in the Modbus telegram	Address in the end device (device-dependent)
Memory[0]	Word	16#4000	44001
Memory[1]	Word	16#4001	44002

5 PLC libraries

5.1 Function blocks

5.1.1 ModbusRtuMaster_PcCom



The function block *ModbusRtuMaster_PcCom* implements a Modbus master that communicates via a serial PC interface (COM port). The block is not called in its basic form, but individual actions of that block are used within a PLC program. Each Modbus function is implemented as an action.

The function block ModbusRtuMaster_KL6x5B [▶ 21] is available for communication via a serial Bus Terminal KL6001, KL6011 or KL6021.

Supported Modbus functions (actions)

- **ModbusMaster.ReadCoils**
  Modbus function 1 = *Read Coils*

  Reads binary outputs (coils) from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.ReadInputStatus**
  Modbus function 2 = *Read Input Status*

  Reads binary inputs from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.ReadRegs**
  Modbus function 3 = *Read Holding Registers*

  Reads data from a connected slave.
- **ModbusMaster.ReadInputRegs**
  Modbus function 4 = *Read Input Registers*

  Reads input registers from a connected slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus function 5 = *Write Single Coil*
  Sends a binary output (coil) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteSingleRegister**
  Modbus function 6 = *Write Single Register*
  Sends a single data word to a connected slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus function 15 = *Write Multiple Coils*
  Sends binary outputs (coils) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteRegs**
  Modbus function 16 = *Preset Multiple Registers*
  Sends data to a connected slave

- **ModbusMaster.Diagnostics**
  Modbus function 8 = *Diagnostics*
  Sends a diagnostic request to the slave with a user-specified function code (subfunction code). Since this function does not address a memory, the function code is transferred in the data word *MBAddr*. Any data required for the function is included in *pMemoryAddr*.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
Quantity: WORD;
MBAddr : WORD;
cbLength: UINT;
pMemoryAddr : DWORD;
Execute : BOOL;
Timeout : TIME;
END_VAR
```

UnitID [▸ 28]

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

Quantity: Number of data words to be read or written for word-oriented Modbus functions. For bit-oriented Modbus functions, Quantity specifies the number of bits (inputs or coils).

MBAddr: Modbus data address, from which the data are read from the end device (slave). This address is transferred to the slave unchanged and interpreted as a data address.
In the *Diagnostics* function (8), the function code (subfunction code) is transferred here.

cbLength : Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the transferred data quantity as specified by Quantity. Example for word access: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated via SIZEOF (Modbus data).

pMemoryAddr: Memory address in the PLC, calculated with ADR (Modbus data). For read actions, the read data are stored in the addressed variable. For send actions, the data are transferred from the addressed variable to the end device.

Execute : Start signal. The action is initiated via a rising edge at the Execute input.

Timeout: Timeout value for waiting for a response from the addressed slave.

VAR_OUTPUT

```
VAR_OUTPUT
BUSY : BOOL;
Error : BOOL;
ErrorId : MODBUS_ERRORS;
cbRead : UINT;
END_VAR
```

Busy: Indicates that the function block is active. *Busy* becomes TRUE with a rising edge at *Execute* and becomes FALSE again once the started action is completed. At any one time, only one action can be active.

Error: Indicates that an error occurred during execution of an action.

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

cbRead: Provides the number of read data bytes for a read action

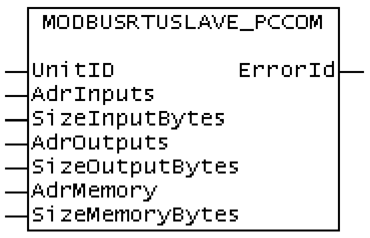
Hardware connection

The data structures required for the link with the communication port are included in the function block. They are displayed in the TwinCAT System Manager once the PLC program has been integrated and can be connected with a COM port. The procedure is as described in Chapter Serial PC Interface.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

5.1.2 ModbusRtuSlave_PcCom



The function block *ModbusRTUslave_PcCom* implements a Modbus slave that communicates via a serial PC interface (COM port). The block is passive until it receives telegrams from a connected Modbus master.

An example program [▸ 32] explains the operating principle.

The function block ModbusRTUslave_KL6x5B [▸ 23] is available for communication via a serial Bus Terminal KL6001, KL6011 or KL6021.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
AdrInputs : POINTER TO BYTE; (* Pointer to the Modbus input area *)
SizeInputBytes : UINT;
AdrOutputs : POINTER TO BYTE; (* Pointer to the Modbus output area *)
SizeOutputBytes : UINT;
AdrMemory : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
SizeMemoryBytes : UINT;
END_VAR
```

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

AdrInputs: Start address of the Modbus input array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (input variable).

SizeInputBytes: Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).

AdrOutputs : Start address of the Modbus output array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (output variable).

SizeOutputBytes: Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable.

AdrMemory : Start address of the Modbus memory array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).

SizeMemoryBytes : Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

VAR_OUTPUT

```
VAR_OUTPUT
ErrorId : Modbus_ERRORS;
END_VAR
```

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

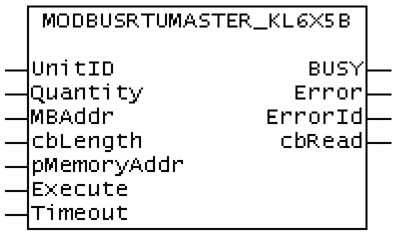
Hardware connection

The data structures required for the link with the communication port are included in the function block. They are displayed in the TwinCAT System Manager once the PLC program has been integrated and can be connected with a COM port. The procedure is as described in Chapter Serial PC Interface.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

5.1.3 ModbusRtuMaster_KL6x5B



The function block *ModbusRtuMaster_KL6x5B* implements a Modbus master that communicates via a serial Bus Terminal KL6001, KL6011 or KL6021. The block is not called in its basic form, but individual actions of that block are used within a PLC program. Each Modbus function is implemented as an action.

Supported Modbus functions (actions)

- **ModbusMaster.ReadCoils**
  Modbus function 1 = *Read Coils*

  Reads binary outputs (coils) from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.

- **ModbusMaster.ReadInputStatus**
  Modbus function 2 = *Read Input Status*

  Reads binary inputs from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.

- **ModbusMaster.ReadRegs**
  Modbus function 3 = *Read Holding Registers*

  Reads data from a connected slave.

- **ModbusMaster.ReadInputRegs**
  Modbus function 4 = *Read Input Registers*

Reads input registers from a connected slave.

- **ModbusMaster.WriteSingleCoil**
  Modbus function 5 = *Write Single Coil*
  Sends a binary output (coil) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteSingleRegister**
  Modbus function 6 = *Write Single Register*
  Sends a single data word to a connected slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus function 15 = *Write Multiple Coils*
  Sends binary outputs (coils) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteRegs**
  Modbus function 16 = *Preset Multiple Registers*
  Sends data to a connected slave
- **ModbusMaster.Diagnostics**
  Modbus function 8 = *Diagnostics*
  Sends a diagnostic request to the slave with a user-specified function code (subfunction code). Since this function does not address a memory, the function code is transferred in the data word *MBAddr*.
  Any data required for the function is included in *pMemoryAddr*.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
Quantity: WORD;
MBAddr : WORD;
cbLength: UINT;
pMemoryAddr : DWORD;
Execute : BOOL;
Timeout : TIME;
END_VAR
```

UnitID [▸ 28]

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

Quantity: Number of data words to be read or written for word-oriented Modbus functions. For bit-oriented Modbus functions, Quantity specifies the number of bits (inputs or coils).

MBAddr: Modbus data address, from which the data are read from the end device (slave). This address is transferred to the slave unchanged and interpreted as a data address.
In the *Diagnostics* function (8), the function code (subfunction code) is transferred here.

cbLength : Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the transferred data quantity as specified by Quantity. Example for word access: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated via SIZEOF (Modbus data).

pMemoryAddr: Memory address in the PLC, calculated with ADR (Modbus data). For read actions, the read data are stored in the addressed variable. For send actions, the data are transferred from the addressed variable to the end device.

Execute : Start signal. The action is initiated via a rising edge at the Execute input.

Timeout: Timeout value for waiting for a response from the addressed slave.

VAR_OUTPUT

```
VAR_OUTPUT
BUSY : BOOL;
Error : BOOL;
ErrorId : MODBUS_ERRORS;
```

```
cbRead : UINT;
ND_VAR
```

Busy: Indicates that the function block is active. *Busy* becomes TRUE with a rising edge at *Execute* and becomes FALSE again once the started action is completed. At any one time, only one action can be active.

Error: Indicates that an error occurred during execution of an action.

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

cbRead: Provides the number of read data bytes for a read action

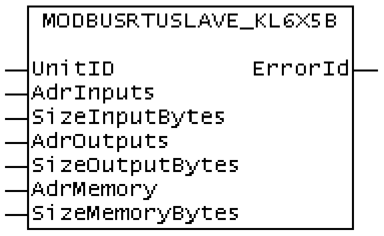
Hardware connection

The data structures required for the link with the communication port are included in the function block. The allocation in the TwinCAT System Manager on a PC is carried out according to the description in Chapter Serial Bus Terminal. On a BC Bus Controller, the I/O addresses have to be assigned manually. See Hardware assignment at the BC Bus Controller.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

5.1.4 ModbusRtuSlave_KL6x5B



The function block *ModbusRtuSlave_KL6x5B* implements a Modbus slave that communicates via a serial Bus Terminal KL6001, KL6011 or KL6021. The block is passive until it receives telegrams from a connected Modbus master.

An example program for a Bus Controller (BC) illustrates the functionality.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
AdrInputs : POINTER TO BYTE; (* Pointer to the Modbus input area *)
SizeInputBytes : UINT;
AdrOutputs : POINTER TO BYTE; (* Pointer to the Modbus output area *)
SizeOutputBytes : UINT;
AdrMemory : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
SizeMemoryBytes : UINT;
END_VAR
```

UnitID [▸ 28]

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

AdrInputs: Start address of the Modbus input array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (input variable).

SizeInputBytes: Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).

AdrOutputs : Start address of the Modbus output array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (output variable).

SizeOutputBytes: Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable.

AdrMemory : Start address of the Modbus memory array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).

SizeMemoryBytes : Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

VAR_OUTPUT

```
VAR_OUTPUT
ErrorId : Modbus_ERRORS;
ND_VAR
```

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

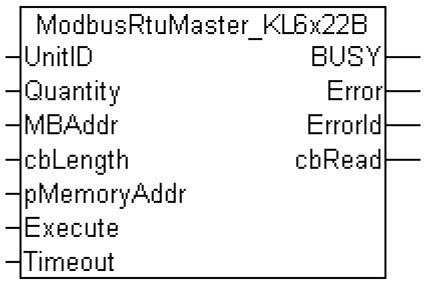
Hardware connection

The data structures required for the link with the communication port are included in the function block. The allocation in the TwinCAT System Manager on a PC is carried out according to the description in Chapter Serial Bus Terminal. On a BC Bus Controller, the I/O addresses have to be assigned manually. See Hardware assignment at the BC Bus Controller.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

5.1.5 ModbusRtuMaster_KL6x22B



The function block *ModbusRtuMaster_KL6x22B* realizes a Modbus master, which communicates via a serial Bus Terminal KL6031 or KL6041. The function block is not called in its basic form, but individual actions of that block are used within a PLC program. Each Modbus function is implemented as an action.

The function block ModbusRtuMaster_PcCom [▸ 18] is available for communication via a serial PC Interface (COM port).

Supported Modbus functions (actions)

- **ModbusMaster.ReadCoils**
  Modbus function 1 = *Read Coils*

    Reads binary outputs (coils) from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.ReadInputStatus**
  Modbus function 2 = *Read Input Status*

    Reads binary inputs from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.ReadRegs**
  Modbus function 3 = *Read Holding Registers*

    Reads data from a connected slave.
- **ModbusMaster.ReadInputRegs**
  Modbus function 4 = *Read Input Registers*

    Reads input registers from a connected slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus function 5 = *Write Single Coil*
  Sends a binary output (coil) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteSingleRegister**
  Modbus function 6 = *Write Single Register*
  Sends a single data word to a connected slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus function 15 = *Write Multiple Coils*
  Sends binary outputs (coils) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteRegs**
  Modbus function 16 = *Preset Multiple Registers*
  Sends data to a connected slave
- **ModbusMaster.Diagnostics**
  Modbus function 8 = *Diagnostics*
  Sends a diagnostic request to the slave with a user-specified function code (subfunction code). Since this function does not address a memory, the function code is transferred in the data word *MBAddr*. Any data required for the function is included in *pMemoryAddr*.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
Quantity: WORD;
MBAddr : WORD;
cbLength: UINT;
pMemoryAddr : DWORD;
Execute : BOOL;
Timeout : TIME;
END_VAR
```

UnitID [▸ 28]

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

Quantity: Number of data words to be read or written for word-oriented Modbus functions. For bit-oriented Modbus functions, Quantity specifies the number of bits (inputs or coils).

MBAddr: Modbus data address, from which the data are read from the end device (slave). This address is transferred to the slave unchanged and interpreted as a data address.
In the *Diagnostics* function (8), the function code (subfunction code) is transferred here.

cbLength : Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the transferred data quantity as specified by Quantity. Example for word access: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated via SIZEOF (Modbus data).

pMemoryAddr: Memory address in the PLC, calculated with ADR (Modbus data). For read actions, the read data are stored in the addressed variable. For send actions, the data are transferred from the addressed variable to the end device.

Execute : Start signal. The action is initiated via a rising edge at the Execute input.

Timeout: Timeout value for waiting for a response from the addressed slave.

VAR_OUTPUT

```
VAR_OUTPUT
BUSY : BOOL;
Error : BOOL;
ErrorId : MODBUS_ERRORS;
cbRead : UINT;
ND_VAR
```

Busy: Indicates that the function block is active. *Busy* becomes TRUE with a rising edge at *Execute* and becomes FALSE again once the started action is completed. At any one time, only one action can be active.

Error: Indicates that an error occurred during execution of an action.

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

cbRead: Provides the number of read data bytes for a read action

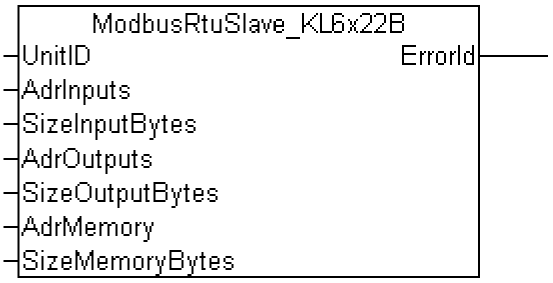
Hardware connection

The data structures required for the link with the communication port are included in the function block. The allocation in the TwinCAT System Manager on a PC is carried out according to the description in Chapter Serial Bus Terminal. On a BC Bus Controller, the I/O addresses have to be assigned manually. See Hardware assignment at the BC Bus Controller.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

5.1.6 ModbusRtuSlave_KL6x22B



The function block *ModbusRTUslave_KL6x22B* realizes a Modbus slave, which communicates via a serial Bus Terminal KL6031 or KL6041. The block is passive until it receives telegrams from a connected Modbus master.

An example program for PC or CX1000 or for a BC Bus Controller explains the operating principle.

The function block ModbusRTUslave_PcCom [▸ 20] is available for communication via a serial PC Interface (COM port).

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
AdrInputs : POINTER TO BYTE; (* Pointer to the Modbus input area *)
SizeInputBytes : UINT;
AdrOutputs : POINTER TO BYTE; (* Pointer to the Modbus output area *)
SizeOutputBytes : UINT;
AdrMemory : POINTER TO BYTE; (* Pointer to the Modbus memory area *)
SizeMemoryBytes : UINT;
END_VAR
```

UnitID [▸ 28]

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

AdrInputs: Start address of the Modbus input array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (input variable).

SizeInputBytes: Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).

AdrOutputs : Start address of the Modbus output array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (output variable).

SizeOutputBytes: Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable.

AdrMemory : Start address of the Modbus memory array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).

SizeMemoryBytes : Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

VAR_OUTPUT

```
VAR_OUTPUT
ErrorId : Modbus_ERRORS;
END_VAR
```

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

Hardware connection

The data structures required for the link with the communication port are included in the function block. The allocation in the TwinCAT System Manager on a PC is carried out according to the description in Chapter Serial Bus Terminal. On a BC Bus Controller, the I/O addresses have to be assigned manually. See Hardware assignment at the BC Bus Controller.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

5.2 Datatypes

5.2.1 Modbus station address

Modbus defines valid station addresses in the range 1 to 247. A Modbus slave only responds to telegrams that contain its own address. Address 0 is not a valid station address. It is used for broadcast telegrams to all stations. These are not answered. Addresses 248 to 255 are reserved.

The ModbusRTU library defines further collective addresses. This enables a station to respond to several addresses.

```
TYPE MODBUS_UNITID :
(
MODBUS_UNITID_BROADCAST := 0,
MODBUS_UNITID_ALLVALID := 256, (* response on address 1..247 *)
MODBUS_UNITID_ALLBUTBROADCAST := 257, (* response on address 1..255 *)
MODBUS_UNITID_ALL := 258 (* response on address 0..255 *)
);
END_TYPE
```

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

5.3 Global Constants

5.3.1 Library Version

All libraries have a certain version. The version is indicated in the PLC library repository, for example. A global constant contains the information about the library version:

Global_Version

```
VAR_GLOBAL CONSTANT
stLibVersion_Tc2_Modbus_RTU : ST_LibVersion;
END_VAR
```

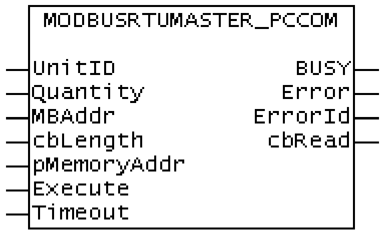
To check whether the version you have is the version you need, use the function F_CmpLibVersion (defined in Tc2_System library).

Notice: All other options for comparing library versions, which you may know from TwinCAT 2, are outdated!

6 Samples

6.1 Modbus RTU Master PC COM port

The function block *ModbusRtuMaster_PcCom* implements a Modbus master that communicates via a serial PC interface (COM port). The function block is not called in its basic form, but individual actions of that block are used within a PLC program. Each Modbus function is implemented as an action.



The function block ModbusRtuMaster_KL6x5B [▶ 21] is available for communication via a serial Bus Terminal KL6001, KL6011 or KL6021.

Supported Modbus functions (actions)

- **ModbusMaster.ReadCoils**
  Modbus function 1 = *Read Coils*

  Reads binary outputs (coils) from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.ReadInputStatus**
  Modbus function 2 = *Read Input Status*

  Reads binary inputs from a connected slave. The data is stored in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.ReadRegs**
  Modbus function 3 = *Read Holding Registers*

  Reads data from a connected slave.
- **ModbusMaster.ReadInputRegs**
  Modbus function 4 = *Read Input Registers*

  Reads input registers from a connected slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus function 5 = *Write Single Coil*
  Sends a binary output (coil) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteSingleRegister**
  Modbus function 6 = *Write Single Register*
  Sends a single data word to a connected slave
- **ModbusMaster.WriteMultipleCoils**
  Modbus function 15 = *Write Multiple Coils*
  Sends binary outputs (coils) to a connected slave. The data must be ready to send in compressed form (8 bits per byte) from the specified address *pMemoryAddr*.
- **ModbusMaster.WriteRegs**
  Modbus function 16 = *Preset Multiple Registers*
  Sends data to a connected slave
- **ModbusMaster.Diagnostics**
  Modbus function 8 = *Diagnostics*
  Sends a diagnostic request to the slave with a user-specified function code (subfunction code). Since this function does not address a memory, the function code is transferred in the data word *MBAddr*. Any data required for the function is included in *pMemoryAddr*.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
Quantity: WORD;
MBAddr : WORD;
cbLength: UINT;
pMemoryAddr : DWORD;
Execute : BOOL;
Timeout : TIME;
END_VAR
```

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

Quantity: Number of data words to be read or written for word-oriented Modbus functions. For bit-oriented Modbus functions, Quantity specifies the number of bits (inputs or coils).

MBAddr: Modbus data address, from which the data are read from the end device (slave). This address is transferred to the slave unchanged and interpreted as a data address.
In the *Diagnostics* function (8), the function code (subfunction code) is transferred here.

cbLength : Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the transferred data quantity as specified by Quantity. Example for word access: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated via SIZEOF (Modbus data).

pMemoryAddr: Memory address in the PLC, calculated with ADR (Modbus data). For read actions, the read data are stored in the addressed variable. For send actions, the data are transferred from the addressed variable to the end device.

Execute : Start signal. The action is initiated via a rising edge at the Execute input.

Timeout: Timeout value for waiting for a response from the addressed slave.

VAR_OUTPUT

```
VAR_OUTPUT
BUSY : BOOL;
Error : BOOL;
ErrorId : MODBUS_ERRORS;
cbRead : UINT;
END_VAR
```

Busy: Indicates that the function block is active. *Busy* becomes TRUE with a rising edge at *Execute* and becomes FALSE again once the started action is completed. At any one time, only one action can be active.

Error: Indicates that an error occurred during execution of an action.

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

cbRead: Provides the number of read data bytes for a read action.

Hardware connection

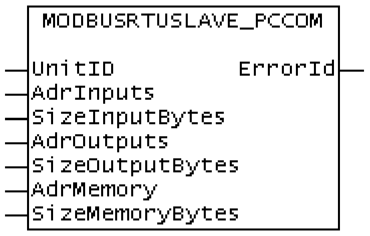
The data structures required for the link with the communication port are included in the function block. They are displayed in the TwinCAT System Manager once the PLC program has been integrated and can be connected with a COM port. The procedure is as described in Chapter Serial PC Interface.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

6.2 Modbus RTU slave via PC COM port

The function block *ModbusRTUslave_PcCom* implements a Modbus slave that communicates via a serial PC interface (COM port). The block is passive until it receives telegrams from a connected Modbus master.



The function block ModbusRTUslave_KL6x5B [▸ 23] is available for communication via a serial Bus Terminal KL6001, KL6011 or KL6021.

VAR_INPUT

```
VAR_INPUT
UnitID : UINT;
AdrInputs : POINTER TO BYTE;  (* Pointer to the Modbus input area *)
SizeInputBytes : UINT;
AdrOutputs : POINTER TO BYTE;  (* Pointer to the Modbus output area *)
SizeOutputBytes : UINT;
AdrMemory : POINTER TO BYTE;  (* Pointer to the Modbus memory area *)
SizeMemoryBytes : UINT;
END_VAR
```

UnitID: Modbus station address [▸ 28] (1..247). The Modbus slave will only answer if it receives telegrams containing its own station address. Optionally, collective addresses can be used for replying to any requests. Address 0 is reserved for broadcast telegrams and is therefore not a valid station address.

AdrInputs: Start address of the Modbus input array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (input variable).

SizeInputBytes: Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).

AdrOutputs : Start address of the Modbus output array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (output variable).

SizeOutputBytes: Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable).

AdrMemory : Start address of the Modbus memory array [▸ 15]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).

SizeMemoryBytes : Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

VAR_OUTPUT

```
VAR_OUTPUT
ErrorId : Modbus_ERRORS;
END_VAR
```

ErrorId: Indicates an error number [▸ 34] in the event of disturbed or faulty communication.

Hardware connection

The data structures required for the link with the communication port are included in the function block. They are displayed in the TwinCAT System Manager once the PLC program has been integrated and can be connected with a COM port. The procedure is as described in Chapter Serial PC Interface.

Prerequisites

Development environment	Target platform	PLC libraries to include
TwinCAT V3.0.0	PC or CX (x86)	Tc2_Modbus_RTU

7 Appendix

7.1 Modbus RTU Error Codes

```
TYPE MODBUS_ERRORS :
(
(* Modbus communication errors *)
MODBUSERROR_NO_ERROR, (* 0 *)
MODBUSERROR_ILLEGAL_FUNCTION, (* 1 *)
MODBUSERROR_ILLEGAL_DATA_ADDRESS, (* 2 *)
MODBUSERROR_ILLEGAL_DATA_VALUE, (* 3 *)
MODBUSERROR_SLAVE_DEVICE_FAILURE, (* 4 *)
MODBUSERROR_ACKNOWLEDGE,(* 5 *)
MODBUSERROR_SLAVE_DEVICE_BUSY, (* 6 *)
MODBUSERROR_NEGATIVE_ACKNOWLEDGE, (* 7 *)
MODBUSERROR_MEMORY_PARITY, (* 8 *)
MODBUSERROR_GATEWAY_PATH_UNAVAILABLE, (* A *)
MODBUSERROR_GATEWAY_TARGET_DEVICE_FAILED_TO_RESPOND, (* B *)

(* additional Modbus error definitions *)
MODBUSERROR_CHARREC_TIMEOUT := 16#20, (* 20 hex *)
MODBUSERROR_ILLEGAL_DATA_SIZE, (* 21 hex *)
MODBUSERROR_ILLEGAL_DEVICE_ADDRESS, (* 22 hex *)
MODBUSERROR_ILLEGAL_DESTINATION_ADDRESS,(* 23 hex *)
MODBUSERROR_ILLEGAL_DESTINATION_SIZE, (* 24 hex *)
MODBUSERROR_NO_RESPONSE,(* 25 hex *)

(* Low level communication errors *)
MODBUSERROR_TXBUFFOVERRUN := 102, (* 102 *)
MODBUSERROR_SENDTIMEOUT := 103, (* 103 *)
MODBUSERROR_DATASIZEOVERRUN := 107, (* 107 *)
MODBUSERROR_STRINGOVERRUN := 110, (* 110 *)
MODBUSERROR_INVALIDPOINTER := 120, (* 120 *)
MODBUSERROR_CRC := 150, (* 150 *)

(* High level PLC errors *)
MODBUSERROR_INVALIDMEMORYADDRESS := 232,(* 232 *)
MODBUSERROR_TRANSMITBUFFERTOOSMALL (* 233 *)
);
END_TYPE
```