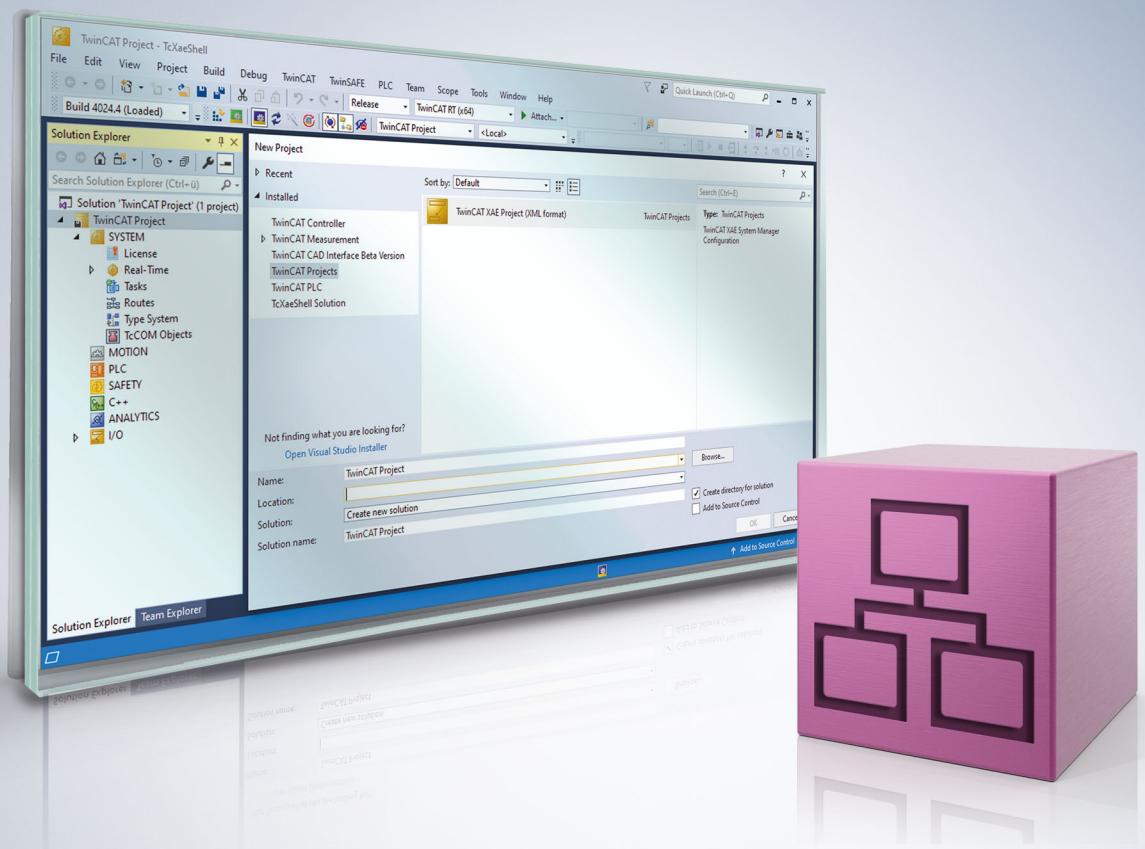
BECKHOFF New Automation Technology

Manual | EN

TF6255

TwinCAT 3 | Modbus RTU



2023-07-19 | Version: 1.4.0

Table of contents

1 Foreword

1.1 Notes on the documentation

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

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

Disclaimer

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

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

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

Copyright

1.2 For your safety

Safety regulations

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

Exclusion of liability

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

Personnel qualification

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

Signal words

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

Personal injury warnings

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

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

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

Warning of damage to property or environment

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

Information on handling the product

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

1.3 Notes on information security

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

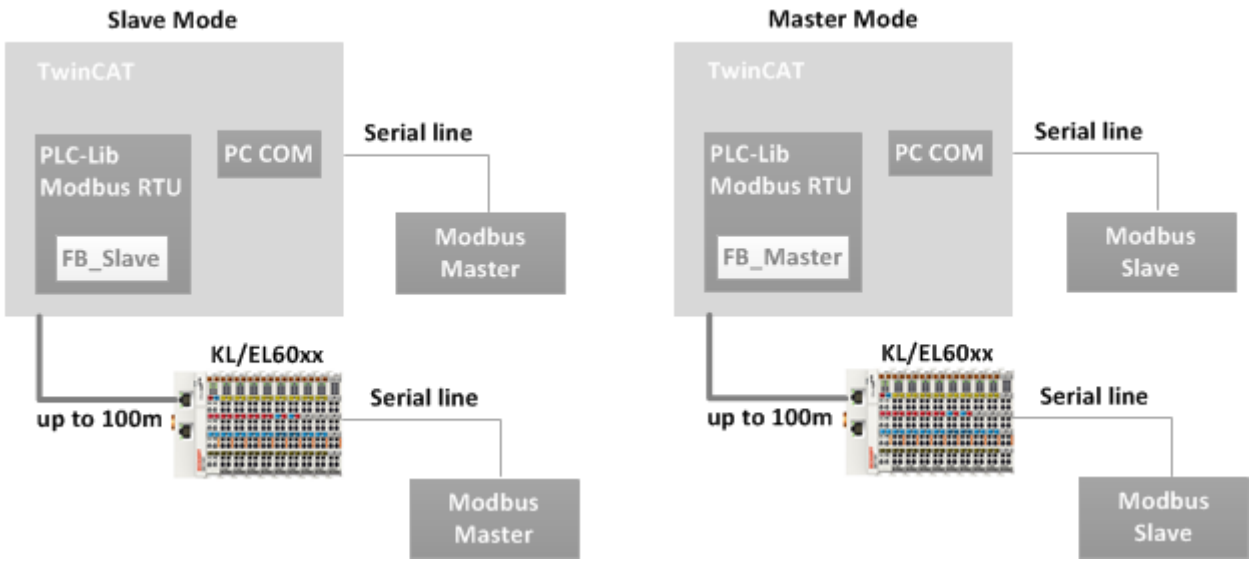
In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

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

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

2 Overview

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



Modbus RTU devices are connected to a Beckhoff Controller via a serial interface. The TwinCAT PLC uses slave function blocks of the Modbus RTU library for communication with my Modbus master (slave mode). In addition, master function blocks are available for addressing several Modbus slaves (master mode)

Supported interfaces

- Serial COM port of a PC or CX
- Serial Bus Terminals KL60xx
- Serial EtherCAT Terminals EL60xx
- Virtual serial COM port (USB port) of a PC or CX
  - With additional use (and licensing) of TF6340 TC3 Serial Communication

Further documentation

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

Boundary conditions

The Modbus protocol defines accurate timing to ensure, for example, the complete transfer of all characters of a telegram. Since the communication Modbus RTU is realized on a PLC controller, accurate timing cannot be guaranteed due to the cyclic execution of the PLC program. Most end devices are very tolerant and function without problems in the event of short time gaps between characters. In individual cases, the behavior of the end device should be checked.

The second channel of an EL60x2 is not suitable for Modbus RTU communication, because it is processed with low priority, which means the frames are sent with gaps, which in turn could be detected by the remote terminal as frame errors.

With some serial interface terminals an internal buffer can be filled before sending (option *continuous sending*). The ModbusRTU library can use this feature if it is set in the corresponding serial terminal. For example, on the KL6031 continuous mode can be activated with the *KL6configuration* configuration function block (register 34 bit 6). Up to 128 bytes are then placed in the internal buffer of the Bus Terminal and transmitted continuously.

3 Installation

3.1 System Requirements

Technical data	TF6255 TC3 Modbus-RTU
Target system	Windows XP / 7 / 10 PC or CX (x86, x64, ARM)
Min. TwinCAT version	3.1.0
Min. TwinCAT level	TC1200 TC3 PLC

3.2 Installation

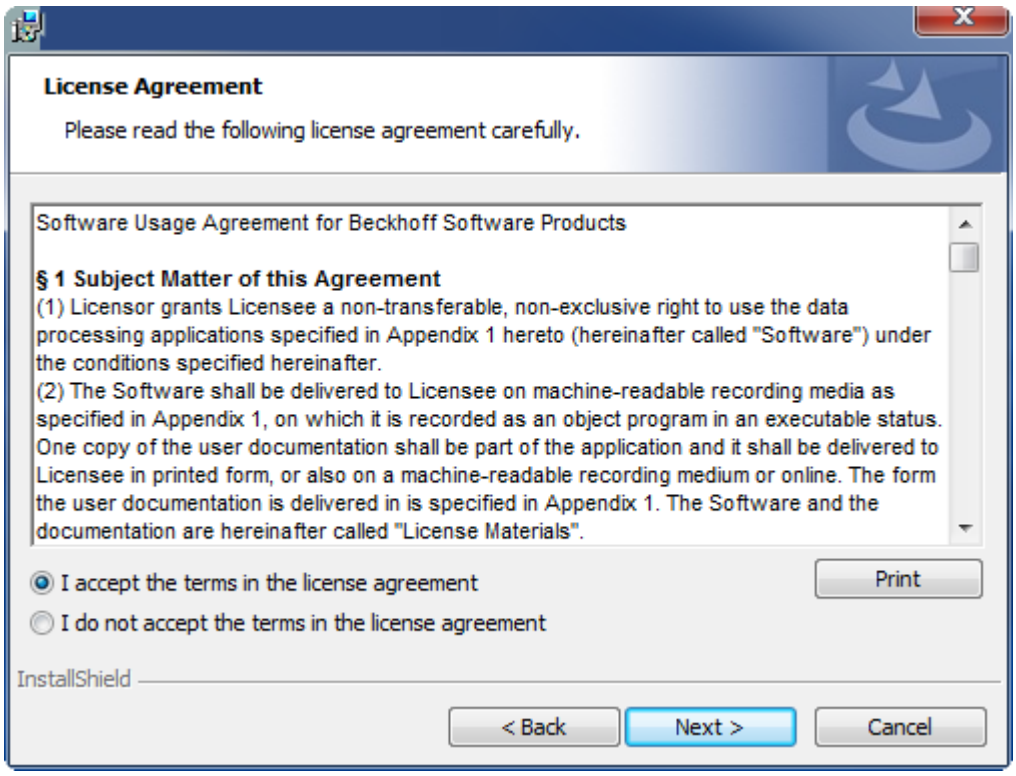
The following section describes how to install the TwinCAT 3 Function for Windows-based operating systems.

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

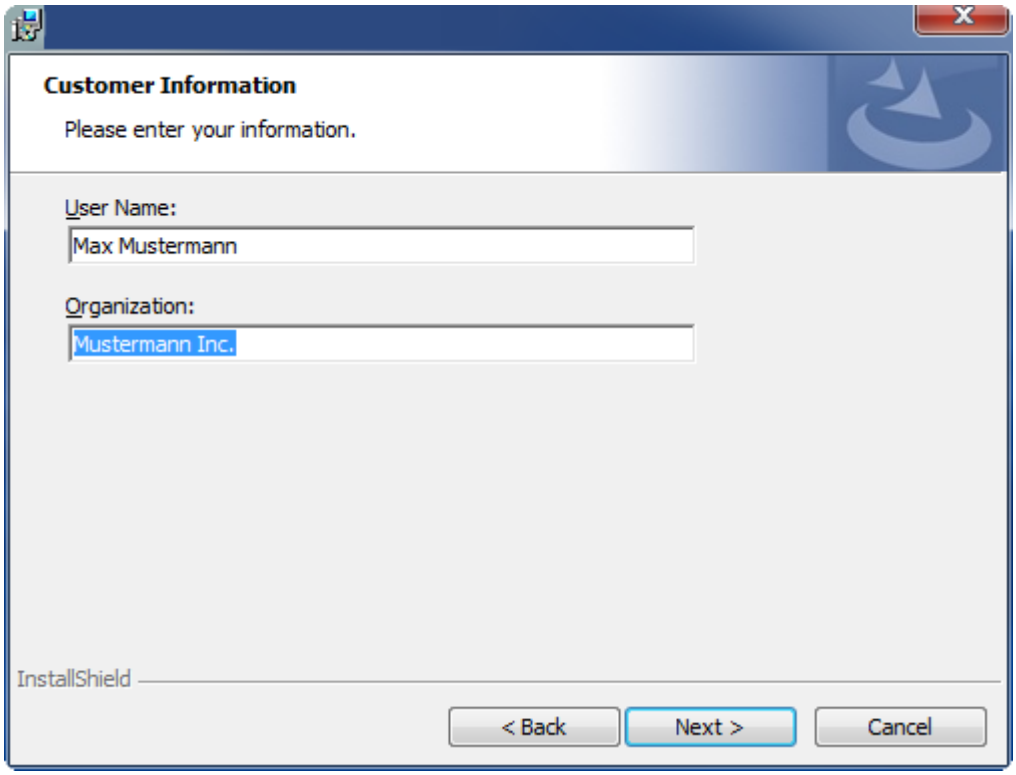
1. Run the setup file as administrator. To do this, select the command **Run as administrator** in the context menu of the file.

   ⇨ The installation dialog opens.

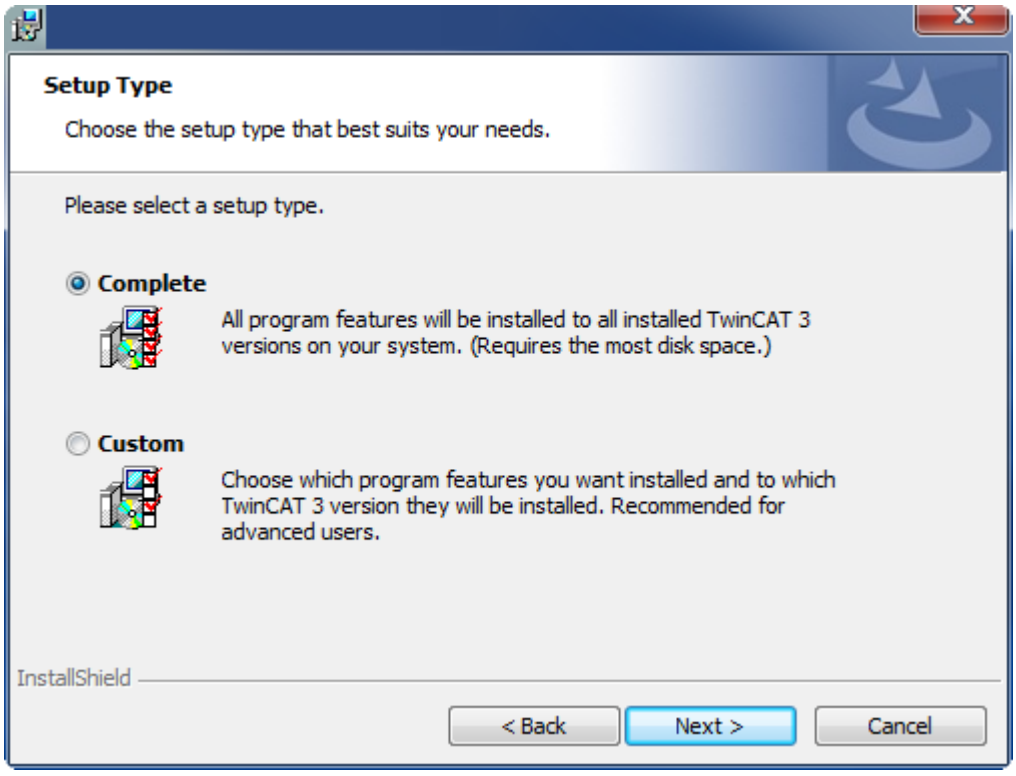
2. Accept the end user licensing agreement and click **Next**.



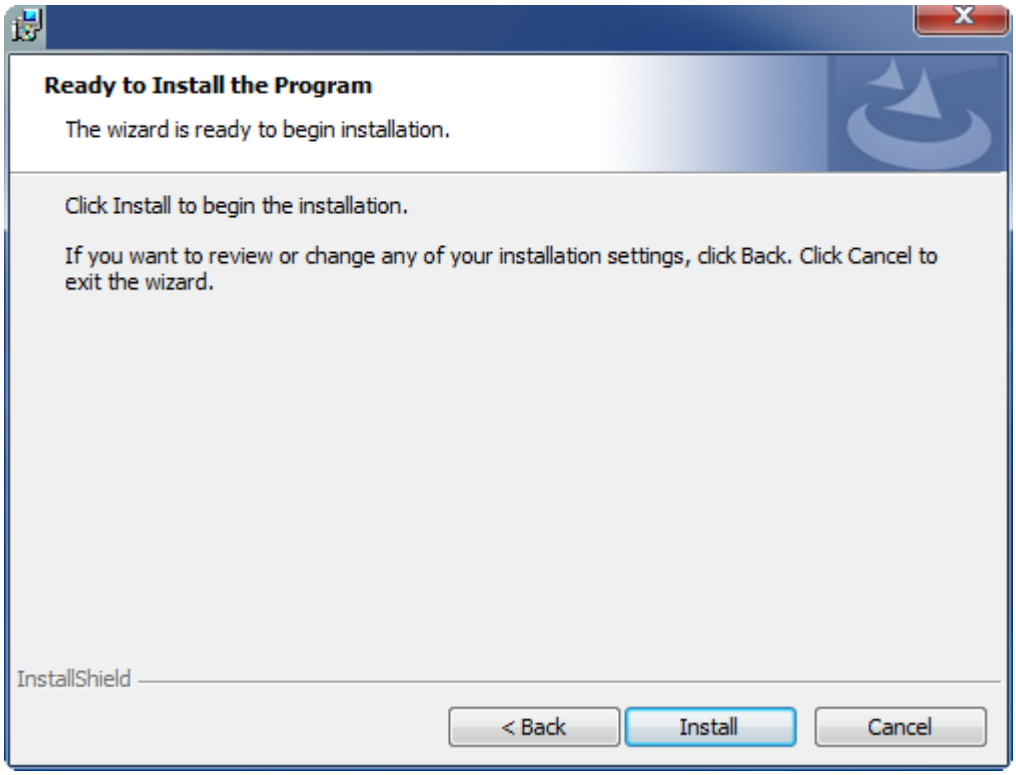
3. Enter your user data.



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**.

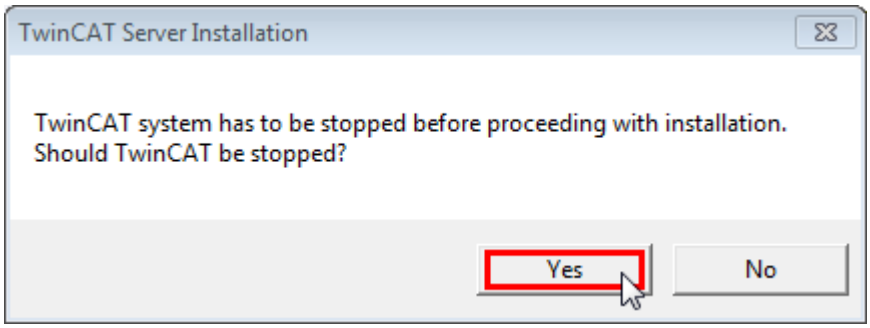


5. Select **Next**, then **Install** to start the installation.

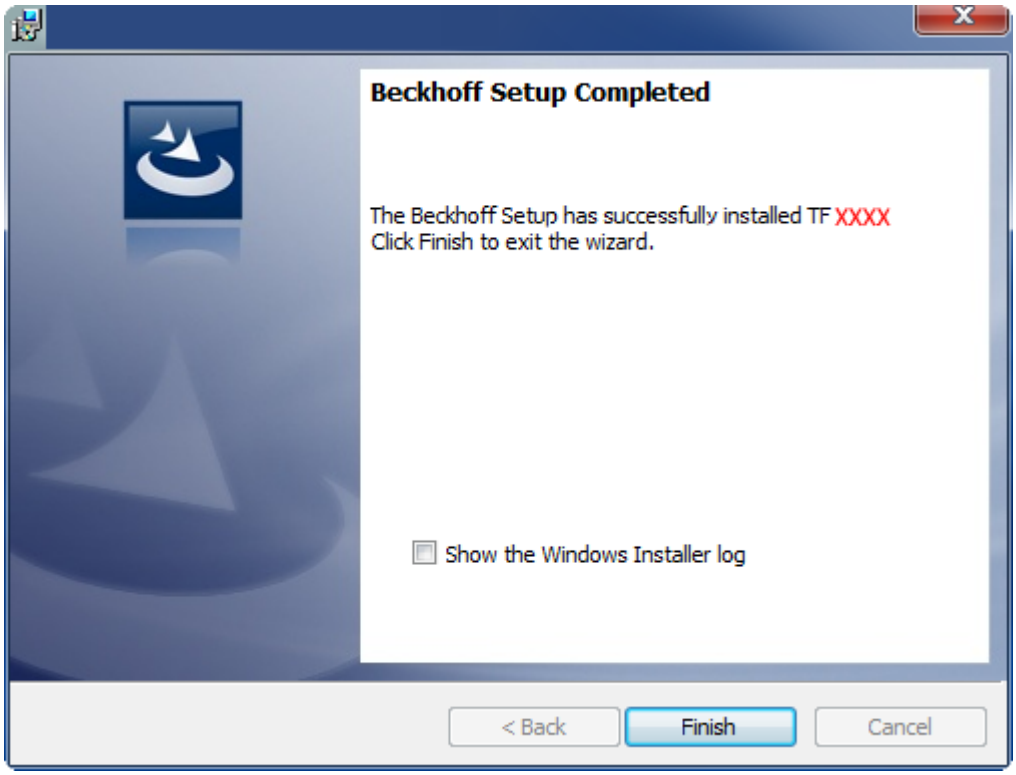


⇨ A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.

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



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



⇨ The TwinCAT 3 Function has been successfully installed and can be licensed (see Licensing [▶ 12]).

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

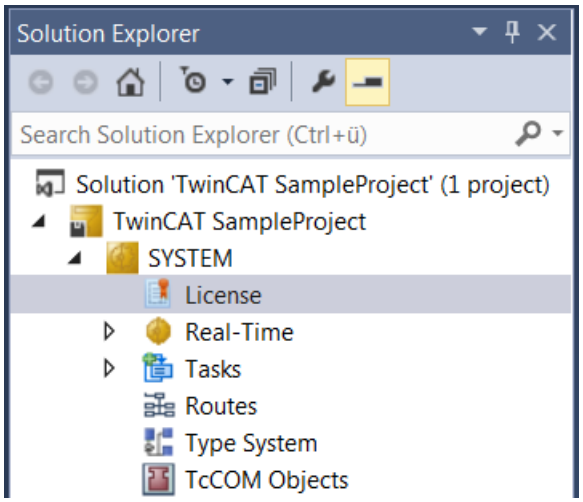
A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

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.

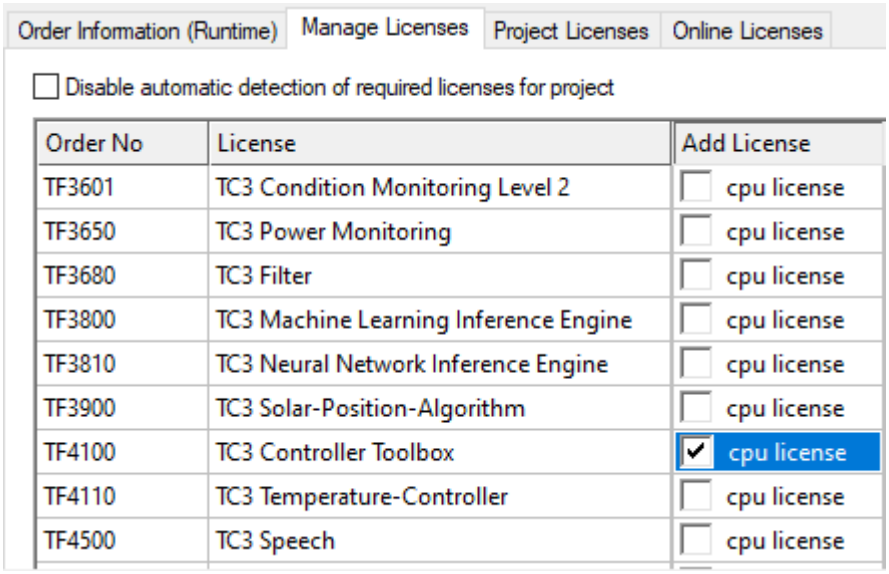
1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the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



⇨ The TwinCAT 3 license manager opens.

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

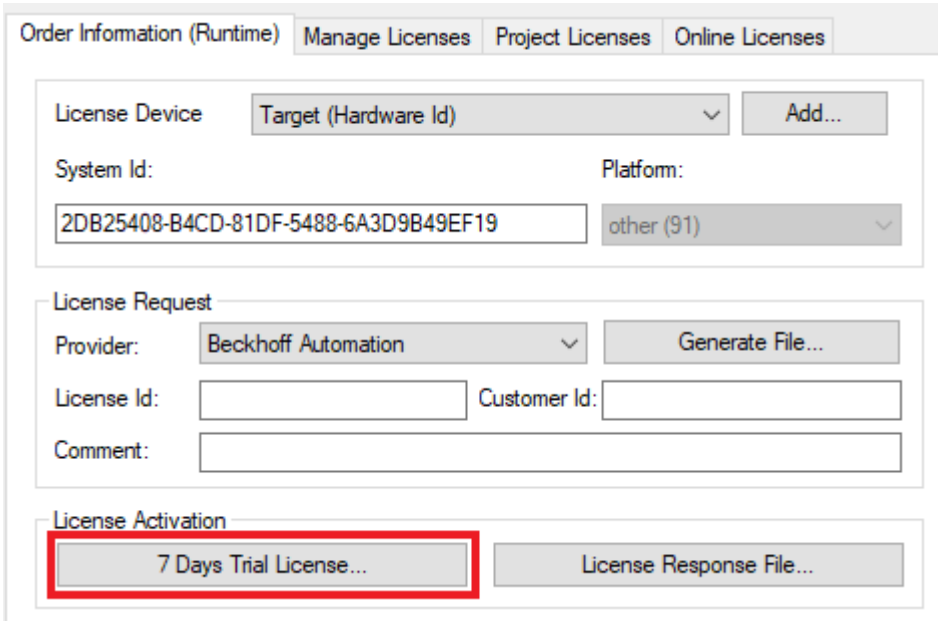


Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu license
TF4100	TC3 Controller Toolbox	cpu license
TF4110	TC3 Temperature-Controller	cpu license
TF4500	TC3 Speech	cpu license

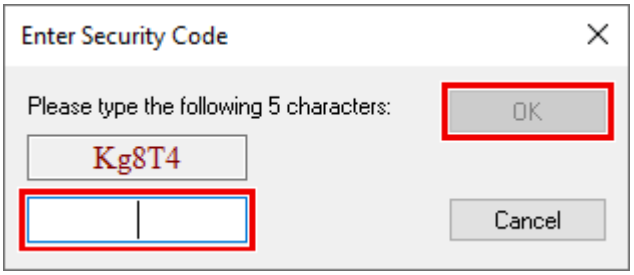
6. Open the **Order Information (Runtime)** tab.

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

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



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



8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.

⇨ The 7-day trial version is enabled.

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 function blocks included in the serial communication library Tc2_SerialCom. This means that the KL6configuration function block can be used license-free for configuring the Bus Terminals.

4.2 Modbus address arrays

Modbus defines access functions for different data areas. These data areas 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 area has a different Modbus start address, so that the areas can be distinguished unambiguously. This offset has to be taken account of for addressing.

Inputs

The *Inputs* data area 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 area 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 area 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 area independent of the physical inputs

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

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

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

Addressing

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

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 area 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 area 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 area 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 area independent of the physical outputs

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

Access to the *Output* area 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* area 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 area.

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 area describes a PLC variable area without physical I/O assignment.

Definition of the Modbus memory data as PLC flags. Start and size of the data area 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* area 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* area is addressed with a 16#4000 offset, i.e. address 16#4000 as transferred in the telegram addresses the first word in the Memory data area.

Examples:

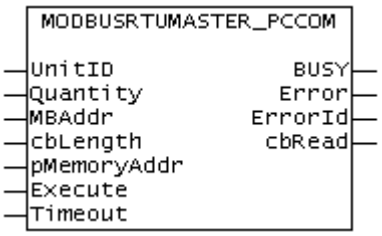
PLC variable	Access type	Address in the Modbus telegram	Address in the end device (device-dependent)
Memory[0]	Word	16#4000	44001
Memory[1]	Word	16#4001	44002

5 PLC API

5.1 Function blocks

5.1.1 [obsolete]

5.1.1.1 ModbusRtuMaster_PcCOM



The function block `ModbusRtuMaster_PcCOM` implements a Modbus master that communicates via a serial PC interface (COM port). The function block ModbusRtuMaster_KL6x5B [▶ 20] is available for communication via a serial Bus Terminal KL6001, KL6011 or KL6021.

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 analogous to the description in the chapter Serial PC Interface of the TF6340 TC3 Serial Communication documentation.

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.

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`.

Inputs

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

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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 (see UnitID [▶ 43])
Quantity	WORD	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	WORD	Modbus data address, from which the data are read from the end device (slave). This address is transferred unchanged to the slave and is interpreted there as data address. With the *Diagnostics* function (8) the function code (subfunction code) is transferred here.
cbLength	UINT	Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the amount of data transferred as determined by Quantity. For word accesses, for example: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated with SIZEOF(Modbus data).
pMemoryAddr	BYTE	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	BOOL	Start signal. The action is triggered by a rising edge at the Execute input.
Timeout	TIME	Timeout value for waiting for a response from the addressed slave.

Outputs

```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
```

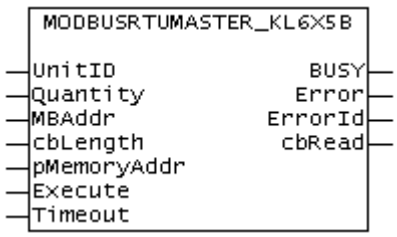
```
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Type	Description
Busy	BOOL	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	BOOL	Indicates that an error occurred during execution of an action.
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.
cbRead	UINT	Provides the number of read data bytes for a read action.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.1.2 ModbusRtuMaster_KL6x5B



The function block `ModbusRtuMaster_KL6x5B` realizes a Modbus master, which communicates via a serial Bus Terminal KL6001, KL6011 or KL6021.

Hardware connection

The data structures required for the link with the communication port are included in the function block. On a PC, the assignment in the TwinCAT System Manager is analogous to the description in the chapter Serial bus terminal of the TF6340 TC3 Serial Communication documentation.

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.

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`.

Inputs

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

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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 (see UnitID [▶ 43])
Quantity	WORD	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	WORD	Modbus data address, from which the data are read from the end device (slave). This address is transferred unchanged to the slave and is interpreted there as data address. With the *Diagnostics* function (8) the function code (subfunction code) is transferred here.
cbLength	UINT	Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the amount of data transferred as determined by Quantity. For word accesses, for example: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated with SIZEOF(Modbus data).
pMemoryAddr	BYTE	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	BOOL	Start signal. The action is triggered by a rising edge at the Execute input.
Timeout	TIME	Timeout value for waiting for a response from the addressed slave.

Outputs

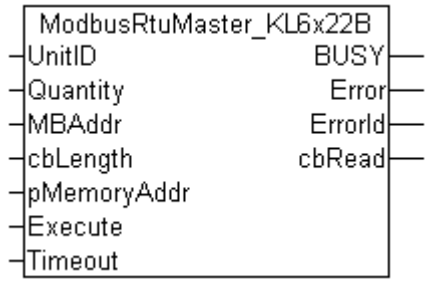
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Type	Description
Busy	BOOL	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	BOOL	Indicates that an error occurred during execution of an action.
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.
cbRead	UINT	Provides the number of read data bytes for a read action.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.1.3 ModbusRtuMaster_KL6x22B



The function block `ModbusRtuMaster_KL6x22B` realizes a Modbus master, which communicates via a serial Bus Terminal KL6031 or KL6041. The function block ModbusRtuMaster_PcCOM [▶ 18] is available for communication via a serial PC interface (COM port).

Hardware connection

The data structures required for the link with the communication port are included in the function block. On a PC, the assignment in the TwinCAT System Manager is analogous to the description in the chapter Serial bus terminal of the TF6340 TC3 Serial Communication documentation.

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.

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`.

Inputs

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

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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 (see UnitID [▶ 43])
Quantity	WORD	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	WORD	Modbus data address, from which the data are read from the end device (slave). This address is transferred unchanged to the slave and is interpreted there as data address. With the *Diagnostics* function (8) the function code (subfunction code) is transferred here.
cbLength	UINT	Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the amount of data transferred as determined by Quantity. For word accesses, for example: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated with SIZEOF(Modbus data).
pMemoryAddr	BYTE	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	BOOL	Start signal. The action is triggered by a rising edge at the Execute input.
Timeout	TIME	Timeout value for waiting for a response from the addressed slave.

Outputs

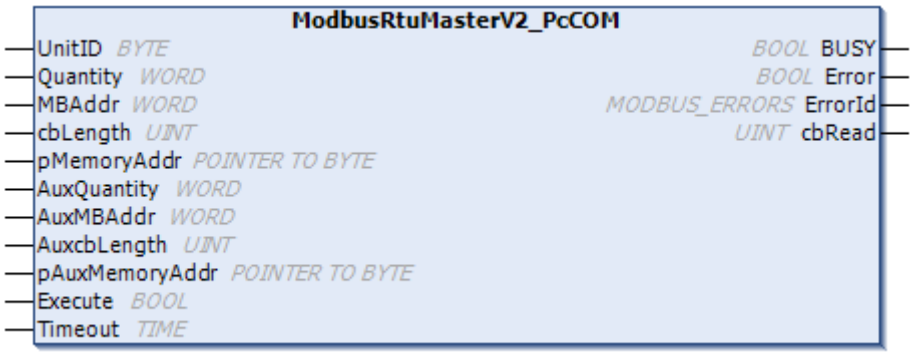
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Type	Description
Busy	BOOL	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	BOOL	Indicates that an error occurred during execution of an action.
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.
cbRead	UINT	Provides the number of read data bytes for a read action.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.2 ModbusRtuMasterV2_PcCOM



The function block `ModbusRtuMasterV2_PcCOM` implements a Modbus master that communicates via a serial PC interface (COM port). Additional function blocks are available for communication via a serial bus terminal.

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 analogous to the description in the chapter Serial PC Interface of the TF6340 TC3 Serial Communication documentation.

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.

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`.

Supported Modbus functions (actions) of the MasterV2 function blocks

- **ModbusMaster.ReadWriteRegs**
  Modbus function 23 = *Read/Write Multiple Registers*

  Sends the data specified via the Aux parameters to a connected slave and receives data from the slave at the same time. The received data is stored at the address specified by pMemoryAddr.
- **ModbusMaster.UserReadWrite**
  Universal user telegram

  The Modbus function code is specified by the user in the first byte of specified data (pMemoryAddr). With this function the user is able to send a Modbus telegram with any function code. Any data received from the slave is stored at the address specified by pAuxMemoryAddr.

Inputs

```
VAR_INPUT
    UnitID       : BYTE;
    Quantity     : WORD;
    MBAddr       : WORD;
    cbLength     : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity     : WORD;
    AuxMBAddr       : WORD;
    AuxcbLength     : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute      : BOOL;
    Timeout      : TIME;
END_VAR
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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 (see UnitID [▶ 43])
Quantity	WORD	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	WORD	Modbus data address, from which the data are read from the end device (slave). This address is transferred unchanged to the slave and is interpreted there as data address. With the *Diagnostics* function (8) the function code (subfunction code) is transferred here.
cbLength	UINT	Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the amount of data transferred as determined by Quantity. For word accesses, for example: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated with SIZEOF(Modbus data).
pMemoryAddr	BYTE	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.
AuxQuantity	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 26]
AuxMBAddr	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 26].
AuxcbLength	UINT	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 26]
pAuxMemoryAddr	BYTE	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 26]
Execute	BOOL	Start signal. The action is triggered by a rising edge at the Execute input.
Timeout	TIME	Timeout value for waiting for a response from the addressed slave.

Outputs

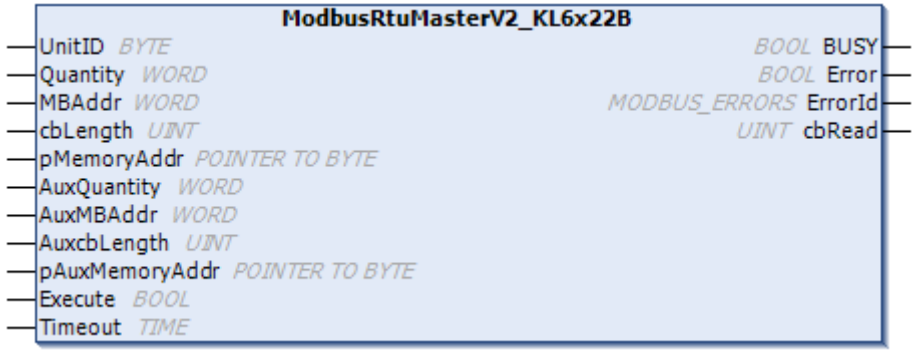
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Type	Description
Busy	BOOL	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	BOOL	Indicates that an error occurred during execution of an action.
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.
cbRead	UINT	Provides the number of read data bytes for a read action.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.3 ModbusRtuMasterV2_KL6x22B



The function block `ModbusRtuMasterV2_KL6x22B` realizes a Modbus master, which communicates via a serial Bus Terminal KL6031 or KL6041. Serial EtherCAT Terminals with 22 bytes of data process image are also supported. The function block ModbusRtuMasterV2_PcCOM [▶ 25] is available for communication via a serial PC interface (COM port).

Hardware connection

The data structures required for the link with the communication port are included in the function block. On a PC, the assignment in the TwinCAT System Manager is analogous to the description in the chapter Serial bus terminal of the TF6340 TC3 Serial Communication documentation.

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.

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`.

Supported Modbus functions (actions) of the MasterV2 function blocks

- **ModbusMaster.ReadWriteRegs**
  Modbus function 23 = *Read/Write Multiple Registers*

  Sends the data specified via the Aux parameters to a connected slave and receives data from the slave at the same time. The received data is stored at the address specified by pMemoryAddr.
- **ModbusMaster.UserReadWrite**
  Universal user telegram

  The Modbus function code is specified by the user in the first byte of specified data (pMemoryAddr). With this function the user is able to send a Modbus telegram with any function code. Any data received from the slave is stored at the address specified by pAuxMemoryAddr.

Inputs

```
VAR_INPUT
    UnitID      : BYTE;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity    : WORD;
    AuxMBAddr      : WORD;
    AuxcbLength    : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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 (see UnitID [▶ 43])
Quantity	WORD	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	WORD	Modbus data address, from which the data are read from the end device (slave). This address is transferred unchanged to the slave and is interpreted there as data address. With the *Diagnostics* function (8) the function code (subfunction code) is transferred here.
cbLength	UINT	Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the amount of data transferred as determined by Quantity. For word accesses, for example: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated with SIZEOF(Modbus data).
pMemoryAddr	BYTE	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.
AuxQuantity	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 29]
AuxMBAddr	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 29].
AuxcbLength	UINT	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 29]
pAuxMemoryAddr	BYTE	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 29]
Execute	BOOL	Start signal. The action is triggered by a rising edge at the Execute input.
Timeout	TIME	Timeout value for waiting for a response from the addressed slave.

Outputs

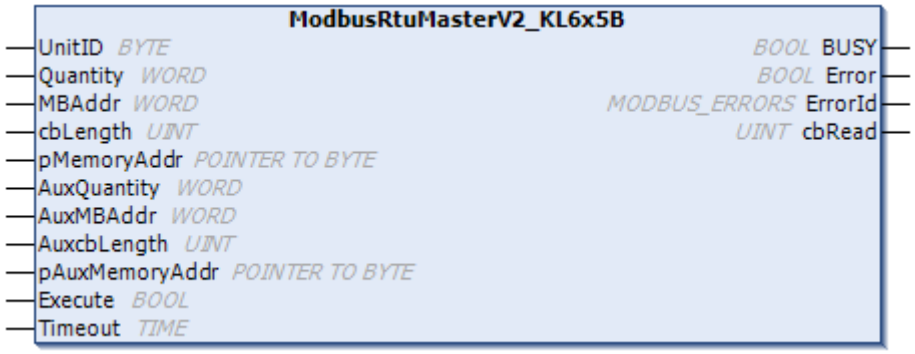
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Type	Description
Busy	BOOL	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	BOOL	Indicates that an error occurred during execution of an action.
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.
cbRead	UINT	Provides the number of read data bytes for a read action.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.4 ModbusRtuMasterV2_KL6x5B



The function block `ModbusRtuMasterV2_KL6x5B` realizes a Modbus master, which communicates via a serial Bus Terminal KL6001, KL6011 or KL6021. The function block ModbusRtuMasterV2_PcCOM [▶ 25] is available for communication via a serial PC interface (COM port).

Hardware connection

The data structures required for the link with the communication port are included in the function block. On a PC, the assignment in the TwinCAT System Manager is analogous to the description in the chapter Serial bus terminal of the TF6340 TC3 Serial Communication documentation.

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.

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`.

Supported Modbus functions (actions) of the MasterV2 function blocks

- **ModbusMaster.ReadWriteRegs**
  Modbus function 23 = *Read/Write Multiple Registers*

  Sends the data specified via the Aux parameters to a connected slave and receives data from the slave at the same time. The received data is stored at the address specified by pMemoryAddr.

- **ModbusMaster.UserReadWrite**
  Universal user telegram

  The Modbus function code is specified by the user in the first byte of specified data (pMemoryAddr). With this function the user is able to send a Modbus telegram with any function code. Any data received from the slave is stored at the address specified by pAuxMemoryAddr.

Inputs

```
VAR_INPUT
    UnitID      : BYTE;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity    : WORD;
    AuxMBAddr      : WORD;
    AuxcbLength    : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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 (see UnitID [▶ 43])
Quantity	WORD	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	WORD	Modbus data address, from which the data are read from the end device (slave). This address is transferred unchanged to the slave and is interpreted there as data address. With the *Diagnostics* function (8) the function code (subfunction code) is transferred here.
cbLength	UINT	Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the amount of data transferred as determined by Quantity. For word accesses, for example: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated with SIZEOF(Modbus data).
pMemoryAddr	BYTE	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.
AuxQuantity	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 32]
AuxMBAddr	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 32].
AuxcbLength	UINT	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 32]
pAuxMemoryAddr	BYTE	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 32]
Execute	BOOL	Start signal. The action is triggered by a rising edge at the Execute input.
Timeout	TIME	Timeout value for waiting for a response from the addressed slave.

Outputs

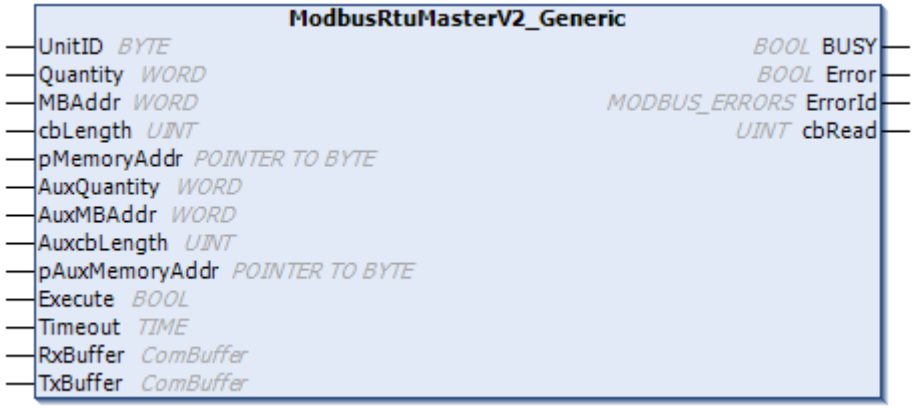
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Type	Description
Busy	BOOL	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	BOOL	Indicates that an error occurred during execution of an action.
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.
cbRead	UINT	Provides the number of read data bytes for a read action.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.5 ModbusRtuMasterV2_Generic



The function block `ModbusRtuMasterV2_Generic` implements a Modbus master that communicates via various serial interfaces (COM port, virtual COM port, EtherCAT Terminals, ...).

Due to the hardware-independent nature of `ModbusRtuMasterV2_Generic`, its use is somewhat more complex than that of the hardware-dependent function blocks `ModbusRtuMasterV2_PcCOM`, `ModbusRtuMasterV2_KL6x22B`, `ModbusRtuMasterV2_ KL6x5B`. All function blocks offer the same ModbusRTU functionality. However, the `ModbusRtuMasterV2_Generic` alone allows the use of virtual COM ports.

Connection to hardware via TF6340 TC3 Serial Communication (license required)

The data structures required to link to the communication port must be instantiated separately. The data structures of type ComBuffer present at this function block are data buffers for decoupling the hardware-dependent background communication. This background communication must be implemented via corresponding function blocks (SerialLineControl, SerialLineControlADS) of the Tc2_SerialCom PLC library, for which this PLC library must first be integrated in the program. The license for TF6340 TC3 Serial Communication is also required.

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.

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`.

Supported Modbus functions (actions) of the MasterV2 function blocks

- **ModbusMaster.ReadWriteRegs**
  Modbus function 23 = *Read/Write Multiple Registers*

  Sends the data specified via the Aux parameters to a connected slave and receives data from the slave at the same time. The received data is stored at the address specified by pMemoryAddr.
- **ModbusMaster.UserReadWrite**
  Universal user telegram

  The Modbus function code is specified by the user in the first byte of specified data (pMemoryAddr). With this function the user is able to send a Modbus telegram with any function code. Any data received from the slave is stored at the address specified by pAuxMemoryAddr.

Inputs

```
VAR_INPUT
    UnitID      : BYTE;
    Quantity    : WORD;
    MBAddr      : WORD;
    cbLength    : UINT;
    pMemoryAddr : POINTER TO BYTE;

    AuxQuantity    : WORD;
    AuxMBAddr      : WORD;
    AuxcbLength    : UINT;
    pAuxMemoryAddr : POINTER TO BYTE;

    Execute     : BOOL;
    Timeout     : TIME;
END_VAR
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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 (see UnitID [▶ 43])
Quantity	WORD	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	WORD	Modbus data address, from which the data are read from the end device (slave). This address is transferred unchanged to the slave and is interpreted there as data address. With the *Diagnostics* function (8) the function code (subfunction code) is transferred here.
cbLength	UINT	Size of the data variable used for send or read actions in bytes. cbLength must be greater than or equal to the amount of data transferred as determined by Quantity. For word accesses, for example: *[cbLength >= Quantity * 2]*. *cbLength* can be calculated with SIZEOF(Modbus data).
pMemoryAddr	BYTE	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.
AuxQuantity	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 35]
AuxMBAddr	WORD	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 35].
AuxcbLength	UINT	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 35]
pAuxMemoryAddr	BYTE	Additional parameter which is only used for read/write functions, see ReadWriteRegs/UserReadWrite [▶ 35]
Execute	BOOL	Start signal. The action is triggered by a rising edge at the Execute input.
Timeout	TIME	Timeout value for waiting for a response from the addressed slave.

Inputs/outputs

```
VAR_IN_OUT
    RxBuffer    : ComBuffer;
    TxBuffer    : ComBuffer;
END_VAR
```

Name	Type	Description
TxBuffer	ComBuffer (Tc2_SerialCom PLC library)	Buffer with send data for the serial hardware being used. This data buffer is never directly written or read by the user, but only serves as a buffer for the communication blocks. The background communication must be realized via corresponding function blocks of the Tc2_SerialCom PLC library.
RxBuffer	ComBuffer (Tc2_SerialCom PLC library)	Buffer into which received data is placed. This data buffer is never directly written or read by the user, but only serves as a buffer for the communication blocks. The background communication must be realized via corresponding function blocks of the Tc2_SerialCom PLC library.

Outputs

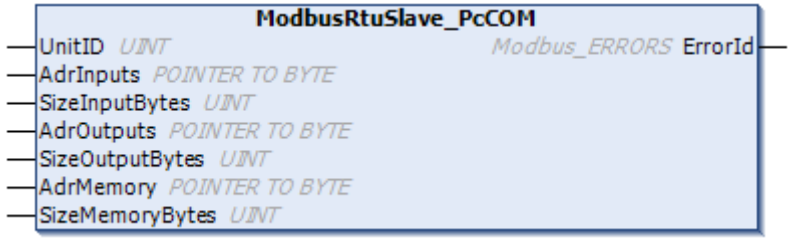
```
VAR_OUTPUT
    BUSY    : BOOL;
    Error   : BOOL;
    ErrorId : MODBUS_ERRORS;
    cbRead  : UINT;
END_VAR
```

Name	Type	Description
Busy	BOOL	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	BOOL	Indicates that an error occurred during execution of an action.
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.
cbRead	UINT	Provides the number of read data bytes for a read action.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU (>= v3.5.6.0)

5.1.6 ModbusRtuSlave_PcCOM



The function block `ModbusRtuSlave_PcCOM` implements a Modbus slave that communicates via a serial PC interface (COM port). The function block ModbusRtuSlave_KL6x5B [▶ 40] is available for communication via a serial Bus Terminal KL6001, KL6011 or KL6021.

The function block is passive until it receives telegrams from a connected Modbus master. A sample program explains the operating principle.

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 analogous to the description in the chapter Serial PC Interface of the TF6340 TC3 Serial Communication documentation.

Inputs

```
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
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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	BYTE	Start address of the Modbus input area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (input variable).
SizeInputBytes	UINT	Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).
AdrOutputs	BYTE	Start address of the Modbus output area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (output variable).
SizeOutputBytes	UINT	Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable).
AdrMemory	BYTE	Start address of the Modbus memory area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).
SizeMemoryBytes	UINT	Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

Outputs

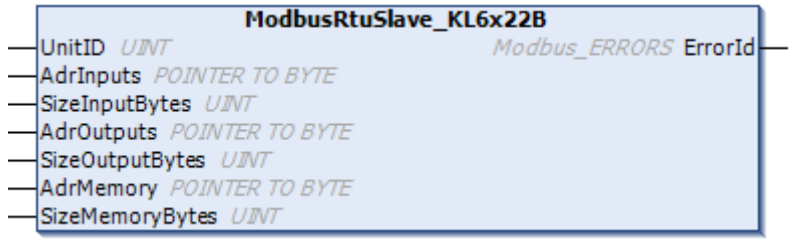
```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Type	Description
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.7 ModbusRtuSlave_KL6x22B



The function block `ModbusRtuSlave_KL6x22B` realizes a Modbus slave, which communicates via a serial Bus Terminal KL6031 or KL6041. Serial EtherCAT Terminals with 22 bytes of data process image are also supported. The function block ModbusRtuSlave_PcCOM [▶ 37] is available for communication via a serial PC interface (COM port).

The function block is passive until it receives telegrams from a connected Modbus master.

Hardware connection

The data structures required for the link with the communication port are included in the function block. On a PC, the assignment in the TwinCAT System Manager is analogous to the description in the chapter Serial bus terminal of the TF6340 TC3 Serial Communication documentation.

Inputs

```
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
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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	BYTE	Start address of the Modbus input area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (input variable).
SizeInputBytes	UINT	Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).
AdrOutputs	BYTE	Start address of the Modbus output area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (output variable).
SizeOutputBytes	UINT	Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable).
AdrMemory	BYTE	Start address of the Modbus memory area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).
SizeMemoryBytes	UINT	Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

Outputs

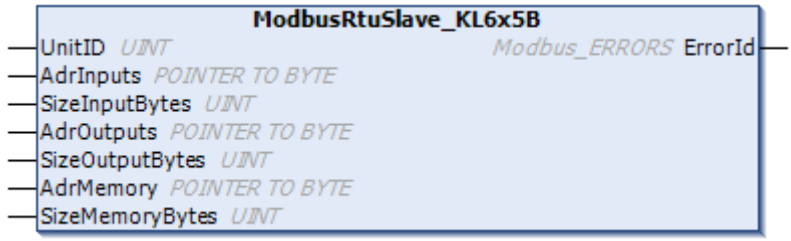
```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Type	Description
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.8 ModbusRtuSlave_KL6x5B



The function block `ModbusRTUslave_KL6x5B` realizes a Modbus slave, which communicates via a serial Bus Terminal KL6001, KL6011 or KL6021. The function block ModbusRtuSlave_PcCOM [▶ 37] is available for communication via a serial PC interface (COM port).

The function block is passive until it receives telegrams from a connected Modbus master.

Hardware connection

The data structures required for the link with the communication port are included in the function block. On a PC, the assignment in the TwinCAT System Manager is analogous to the description in the chapter Serial bus terminal of the TF6340 TC3 Serial Communication documentation.

Inputs

```
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
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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	BYTE	Start address of the Modbus input area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (input variable).
SizeInputBytes	UINT	Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).
AdrOutputs	BYTE	Start address of the Modbus output area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (output variable).
SizeOutputBytes	UINT	Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable).
AdrMemory	BYTE	Start address of the Modbus memory area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).
SizeMemoryBytes	UINT	Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

Outputs

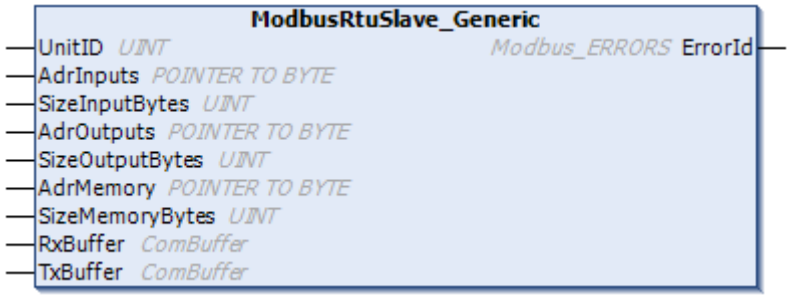
```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Type	Description
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.1.9 ModbusRtuSlave_Generic



The function block `ModbusRtuSlave_Generic` implements a Modbus slave that communicates via various serial interfaces (COM port, virtual COM port, EtherCAT Terminals, ...).

Due to the hardware-independent nature of `ModbusRtuSlave_Generic`, its use is somewhat more complex than that of the hardware-dependent function blocks `ModbusRtuSlave_PcCOM`, `ModbusRtuSlave_KL6x22B`, `ModbusRtuSlave_ KL6x5B`. All function blocks offer the same ModbusRTU functionality. However, the `ModbusRtuSlave_Generic` alone allows the use of virtual COM ports.

The function block is passive until it receives telegrams from a connected Modbus master. A sample program explains the operating principle.

Connection to hardware via TF6340 TC3 Serial Communication (license required)

The data structures required to link to the communication port must be instantiated separately. The data structures of type ComBuffer present at this function block are data buffers for decoupling the hardware-dependent background communication. This background communication must be implemented via corresponding function blocks (SerialLineControl, SerialLineControlADS) of the Tc2_SerialCom PLC library, for which this PLC library must first be integrated in the program. The license for TF6340 TC3 Serial Communication is also required.

Inputs

```
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
```

Name	Type	Description
UnitID	UINT	Modbus station address [▶ 43] (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	BYTE	Start address of the Modbus input area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (input variable).
SizeInputBytes	UINT	Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).
AdrOutputs	BYTE	Start address of the Modbus output area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (output variable).
SizeOutputBytes	UINT	Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable).
AdrMemory	BYTE	Start address of the Modbus memory area [▶ 15]. The data area is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).
SizeMemoryBytes	UINT	Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).

Inputs/outputs

```
VAR_IN_OUT
    RxBuffer    : ComBuffer;
    TxBuffer    : ComBuffer;
END_VAR
```

Name	Type	Description
TxBuffer	ComBuffer (Tc2_SerialCom PLC library)	Buffer with send data for the serial hardware being used. This data buffer is never directly written or read by the user, but only serves as a buffer for the communication blocks. The background communication must be realized via corresponding function blocks of the Tc2_SerialCom PLC library.
RxBuffer	ComBuffer (Tc2_SerialCom PLC library)	Buffer into which received data is placed. This data buffer is never directly written or read by the user, but only serves as a buffer for the communication blocks. The background communication must be realized via corresponding function blocks of the Tc2_SerialCom PLC library.

Outputs

```
VAR_OUTPUT
    ErrorId : MODBUS_ERRORS;
END_VAR
```

Name	Type	Description
ErrorId	MODBUS_ERRORS	Indicates an error number [▶ 44] in the event of disturbed or faulty communication.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4024.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU (>= v3.5.6.0)

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 Tc2_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
```

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.0	PC or CX (x86, x64, ARM)	Tc2_ModbusRTU

5.3 Global Constants

5.3.1 Global_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:

```
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 the Tc2_System PLC library).

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

6 Appendix

6.1 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
```

More Information:
www.beckhoff.com/tf6255

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