Table of contents

1 Foreword ................................................................................................................................. 5
  1.1 Notes on the documentation ............................................................................................... 5
  1.2 Safety instructions ............................................................................................................... 6
2 Overview .................................................................................................................................. 7
3 Installation ................................................................................................................................. 8
  3.1 System Requirements .......................................................................................................... 8
  3.2 Installation ........................................................................................................................... 8
  3.3 Licensing ............................................................................................................................. 11
4 Configuration .......................................................................................................................... 14
  4.1 Terminal configuration ........................................................................................................ 14
  4.2 Modbus address arrays ....................................................................................................... 14
5 PLC libraries ............................................................................................................................ 17
  5.1 Function blocks .................................................................................................................. 17
    5.1.1 ModbusRtuMaster_PcCom ......................................................................................... 17
    5.1.2 ModbusRtuSlave_PcCom .......................................................................................... 19
    5.1.3 ModbusRtuMaster_KL6x5B ....................................................................................... 20
    5.1.4 ModbusRtuSlave_KL6x5B ........................................................................................ 22
    5.1.5 ModbusRtuMaster_KL6x22B ..................................................................................... 23
    5.1.6 ModbusRtuSlave_KL6x22B ....................................................................................... 26
  5.2 Datatypes ............................................................................................................................ 27
    5.2.1 Modbus station address ............................................................................................... 27
  5.3 Global Constants .................................................................................................................. 27
    5.3.1 Library Version ........................................................................................................... 27
6 Samples ..................................................................................................................................... 29
  6.1 Modbus RTU Master PC COM port ..................................................................................... 29
  6.2 Modbus RTU slave via PC COM port ................................................................................... 31
7 Appendix ................................................................................................................................... 33
  7.1 Modbus RTU Error Codes ................................................................................................... 33
1 Foreword

1.1 Notes on the documentation

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

It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.

It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

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

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.

We reserve the right to revise and change the documentation at any time and without prior announcement.

No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.

Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:


with corresponding applications or registrations in various other countries.

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

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.

The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.

Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.
1.2 Safety instructions

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

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

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

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="danger.png" alt="DANGER" /></td>
<td>Serious risk of injury! Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="warning.png" alt="WARNING" /></td>
<td>Risk of injury! Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.</td>
</tr>
<tr>
<td><img src="caution.png" alt="CAUTION" /></td>
<td>Personal injuries! Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.</td>
</tr>
<tr>
<td><img src="note.png" alt="NOTE" /></td>
<td>Damage to the environment or devices Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.</td>
</tr>
</tbody>
</table>

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

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 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

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

<table>
<thead>
<tr>
<th>Technical Data</th>
<th>TF6255 TC3 Modbus-RTU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target System</td>
<td>Windows XP / 7</td>
</tr>
<tr>
<td></td>
<td>PC or CX (x86)</td>
</tr>
<tr>
<td>Min. TwinCAT-Version</td>
<td>3.0.3101</td>
</tr>
<tr>
<td>Min. TwinCAT-Level</td>
<td>TC1200 TC3 PLC</td>
</tr>
</tbody>
</table>

3.2 Installation

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

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

1. Run the setup file as administrator. To do this, select the command Run as administrator in the context menu of the file.
   - The installation dialog opens.
2. Accept the end user licensing agreement and click Next.
3. Enter your user data.

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

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

- A dialog box informs you that the TwinCAT system must be stopped to proceed with the installation.
7. Select **Finish** to exit the setup.

![Beckhoff Setup Completed](image)

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

## 3.3 Licensing

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

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

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

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

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

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

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

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

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

   - The TwinCAT 3 license manager opens.
   - In the tabular overview of licenses, the previously selected license is displayed with the status “missing”.

![Solution Explorer](image-url)

![Manage Licenses](image-url)

![Order Information (Runtime)](image-url)
7. Click **7-Day Trial License**... to activate the 7-day trial license.

![License Activation Dialog](image)

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

![Enter Security Code](image)

- Please type the following 5 characters: Kg8T4

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 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:
### 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.

```plaintext
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

```plaintext
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:

<table>
<thead>
<tr>
<th>PLC variable</th>
<th>Access type</th>
<th>Address in the Modbus telegram</th>
<th>Address in the end device (device-dependent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs[0]</td>
<td>Word</td>
<td>16#800</td>
<td>40801</td>
</tr>
<tr>
<td>Outputs[1]</td>
<td>Word</td>
<td>16#801</td>
<td>40802</td>
</tr>
<tr>
<td>Outputs[0], Bit 0</td>
<td>Bit</td>
<td>16#800</td>
<td>00801</td>
</tr>
<tr>
<td>Outputs[1], Bit 14</td>
<td>Bit</td>
<td>16#81E</td>
<td>0081F</td>
</tr>
</tbody>
</table>

### 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.

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

Definition of the Modbus memory data as variable without flag address
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 memory data array.

Examples:

<table>
<thead>
<tr>
<th>PLC variable</th>
<th>Access type</th>
<th>Address in the Modbus telegram</th>
<th>Address in the end device (device-dependent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory[0]</td>
<td>Word</td>
<td>16#4000</td>
<td>44001</td>
</tr>
<tr>
<td>Memory[1]</td>
<td>Word</td>
<td>16#4001</td>
<td>44002</td>
</tr>
</tbody>
</table>
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 [\[20\]] 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

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

**UnitID** [27]

**UnitID**: Modbus station address [27] (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: \[cbLength \geq Quantity \times 2\]. *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

```plaintext
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 [33] 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>

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 [31] explains the operating principle.

The function block `ModbusRtuSlave_KL6x5B` [22] is available for communication via a serial Bus Terminal KL6001, KL6011 or KL6021.

```plaintext
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 [27] (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 [14]. 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 [14]. 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 [14]. 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 [33]* 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>

### 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

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

**UnitID** [27]

UnitID: Modbus station address (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: \( cbLength \geq Quantity \times 2 \). 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

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

TF6255  Version: 1.2  21
**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 [33] 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>

### 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

```plaintext
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** [27]
UnitID: Modbus station address [27] (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 [14]. 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 [14]. 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 [14]. 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

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorId : Modbus_ERRORS;</td>
</tr>
<tr>
<td>ND_VAR</td>
</tr>
</tbody>
</table>

ErrorId: Indicates an error number [33] 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>

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` 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**

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

**UnitID**

`UnitID` Modbus station address (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: \[\text{cbLength} \geq \text{Quantity} \times 2\]. 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 [33] 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>
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 is available for communication via a serial PC Interface (COM port).

VAR_INPUT

VAR_INPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnitID</td>
<td>Modbus station address (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.</td>
</tr>
<tr>
<td>AdrInputs</td>
<td>Start address of the Modbus input array. The data array is usually declared as a PLC array, and the address can be calculated with ADR (input variable).</td>
</tr>
<tr>
<td>SizeInputBytes</td>
<td>Size of the Modbus input array in bytes. The size can be calculated with SIZEOF (input variable).</td>
</tr>
<tr>
<td>AdrOutputs</td>
<td>Start address of the Modbus output array. The data array is usually declared as a PLC array, and the address can be calculated with ADR (output variable).</td>
</tr>
<tr>
<td>SizeOutputBytes</td>
<td>Size of the Modbus output array in bytes. The size can be calculated with SIZEOF (output variable).</td>
</tr>
<tr>
<td>AdrMemory</td>
<td>Start address of the Modbus memory array. The data array is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).</td>
</tr>
<tr>
<td>SizeMemoryBytes</td>
<td>Size of the Modbus memory array in bytes. The size can be calculated with SIZEOF (memory variable).</td>
</tr>
</tbody>
</table>

VAR_OUTPUT

VAR_OUTPUT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ErrorId</td>
<td>Modbus_ERRORS;</td>
</tr>
</tbody>
</table>
**ErrorId:** Indicates an error number [33] 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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>

### 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.

```plaintext
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**

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>

### 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 [20] 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 [27] (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: \[cbLength \geq Quantity \times 2\]. 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 [33] 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>
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.

```
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 [27] (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 [14]. 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 [14]. 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 [14]. 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
ErrorId : Modbus_ERRORS;
END_VAR
```

**ErrorId**: Indicates an error number [33] 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

<table>
<thead>
<tr>
<th>Development environment</th>
<th>Target platform</th>
<th>PLC libraries to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwinCAT V3.0.0</td>
<td>PC or CX (x86)</td>
<td>Tc2_Modbus_RTU</td>
</tr>
</tbody>
</table>
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
```
More Information:
www.beckhoff.de/tf6255