BECKHOFF New Automation Technology

Manual | EN

TS6255-0030

TwinCAT 2 | Modbus RTU BC

Supplement | Communication

2022-05-09 | Version: 1.0

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

© 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!

⚠ DANGER

Serious risk of injury!

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

NOTE

Damage to the environment or devices

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

Tip or pointer

This symbol indicates information that contributes to better understanding.

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

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

Typical end devices are operating terminals with a Modbus driver that are connected to a TwinCAT controller via a serial RS-232, RS-422 or RS-485 interface. In this case, the TwinCAT PLC is a Modbus slave and the operating terminal is a Modbus master. In this configuration, the programming effort on the PLC side is very low.

Alternatively, Modbus master functions are available in the library, through which the PLC can address one or several Modbus slaves. This configuration is less common and also not really recommended, since the programming effort is higher.

Supported TwinCAT controllers

- TwinCAT PC
- CX1000 series
- Bus controller BC and BX

Supported interfaces

- serial port (COM port) of a PC or CX
- serial port of a BX controller
- serial Bus Terminals KL6001, KL6011, KL6021, KL6031 or KL6041, corresponding EL terminals

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.

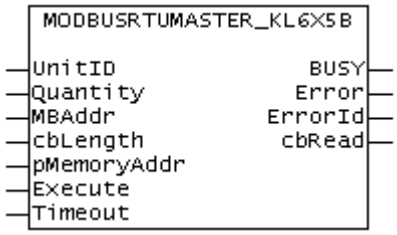
The second channel of an EL60x2 is not suitable for Modbus RTU communication, because it is processed with low priority, which means the frames are sent with gaps, which in turn could be detected by the remote terminal as frame errors.

Note: 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.

Further documentation

http://www.modicon.com/
http://www.modbus.org

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.

An https://infosys.beckhoff.com/content/1033/tcplclibmodbusrtubc/Resources/zip/455297291.zip bus controller explains the operating principle.

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 bit per byte) starting from address *pMemoryAddr*.
- **ModbusMaster.ReadInputStatus**
  Modbus function2 = *Read Input Status*
  Reads binary inputs from a connected slave. The data is stored in compressed form (8 bit per byte) starting from 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 inputs registers from a connected slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus function 5 = *Write Single Coil*
  Sends a binary output (Coil) to a connected slave. The data is supposed to be stored in a compressed form (8 bit pro byte) starting from *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 is supposed to be stored in a compressed form (8 bit pro byte) starting from *address pMemoryAddr*.
- **ModbusMaster.WriteRegs**
  Modbus function 16 = *Preset Multiple Registers*
  Sends data to a connected slave
- **ModbusMaster.Diagnostics**
  Modbus function 8 = *Diagnostics*
  Sends a diagnostics request with a user defined subfunction code to a connected slave. The subfunction code is passed to the function by the *MBAddr* parameter. Additional data can be passed by *pMemoryAddr*.

VAR_INPUT

```
VAR_INPUT
    UnitID     : BYTE;
    Quantity    : WORD;
```

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

UnitID: Modbus Station address [▶ 17] (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.
MBAddr holds a subfuntion code when the *Diagnostics* function is used.

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

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

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

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

VAR_OUTPUT

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

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

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

ErrorId: Indicates an error number [▶ 20] in the event of disturbed or faulty communication.

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

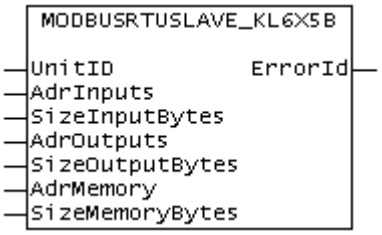
Hardware connection

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 [▶ 21].

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT from V2.8	PC (i386), CX1000	ModbusRTU.lb (Version 2.2 or higher)
TwinCAT from V2.8	Bus controller BC	ModbusRTU.lb6 (Version 2.2 or higher)

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 https://infosys.beckhoff.com/content/1033/tcplclibmodbusrtubc/Resources/zip/455300235.zip explains the operating principle.

VAR_INPUT

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

UnitID : Modbus station address [▶ 17] (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 [▶ 18]. 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 [▶ 18]. 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 [▶ 18]. The data array is usually declared as a PLC array, and the address can be calculated with ADR (memory variable).

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

VAR_OUTPUT

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

ErrorId: Indicates an error number [▶ 20] in the event of disturbed or faulty communication.

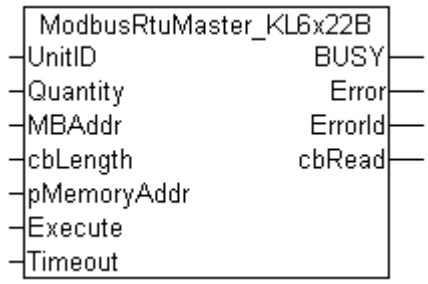
Hardware connection

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 [▶ 21].

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT from V2.8	PC (i386), CX1000,	ModbusRTU.lib
TwinCAT from V2.8	Bus controller BC	ModbusRTU.lb6

5 ModbusRtuMaster_KL6x22B



The function block *ModbusRtuMaster_KL6x22B* implements a Modbus master that communicates via a serial Bus Terminal KL6021 or KL6041. 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.

An example program for a BC bus controller (https://infosys.beckhoff.com/content/1033/tcplclibmodbusrtubc/Resources/zip/455297291.zip) explains the operating principle.

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 bit per byte) starting from address *pMemoryAddr*.
- **ModbusMaster.ReadInputStatus**
  Modbus function2 = *Read Input Status*
  Reads binary inputs from a connected slave. The data is stored in compressed form (8 bit per byte) starting from 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 inputs registers from a connected slave.
- **ModbusMaster.WriteSingleCoil**
  Modbus function 5 = *Write Single Coil*
  Sends a binary output (Coil) to a connected slave . The data is supposed to be stored in a compressed form (8 bit pro byte) starting from *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 is supposed to be stored in a compressed form (8 bit pro byte) starting from *address pMemoryAddr*.
- **ModbusMaster.WriteRegs**
  Modbus function 16 = *Preset Multiple Registers*
  Sends data to a connected slave
- **ModbusMaster.Diagnostics**
  Modbus function 8 = *Diagnostics*
  Sends a diagnostics request with a user defined subfunction code to a connected slave. The subfunction code is passed to the function by the *MBAddr* parameter. Additional data can be passed by *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 (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.
MBAddr holds a subfuntion code when the *Diagnostics* function is used.

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

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

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

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

VAR_OUTPUT

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

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

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

ErrorId: Indicates an error number in the event of disturbed or faulty communication.

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

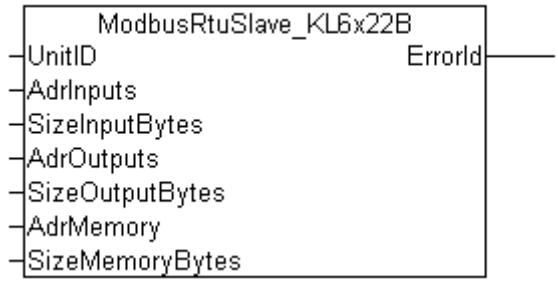
Hardware connection

The data structures required for the link with the communication port are included in the function block. On a BC bus controller, the I/O addresses must be assigned manually. See Hardware assignment at the BC bus controller.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT from V2.8	Bus controller BC	ModbusRTU.lb6 (Version 2.2 or higher)

6 ModbusRtuSlave_KL6x22B



The function block *ModbusRtuSlave_KL6x22B* implements a Modbus slave that 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 a BC bus controller (https://infosys.beckhoff.com/content/1033/tcplclibmodbusrtubc/Resources/zip/455300235.zip) explains the operating principle.

VAR_INPUT

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

UnitID : Modbus station address (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. 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 [▶ 18]. 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 [▶ 19]. 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 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. On a BC bus controller, the I/O addresses must be assigned manually. See Hardware assignment at the BC bus controller.

Requirements

Development environment	Target system type	PLC libraries to be linked
TwinCAT from V2.8	PC (i386), CX1000,	ModbusRTU.lib
TwinCAT from V2.8	Bus controller BC	ModbusRTU.lb6

7 Modbus station address

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

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

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

8 Modbus address arrays

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

Inputs

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

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

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

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

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

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

```
2 : Read input status
    4 : Read Input Registers
```

Addressing

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

Examples:

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

Outputs

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

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

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

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

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

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

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

Addressing

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

Examples:

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

Memory

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

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

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

- Definition of the Modbus memory data as variable without flag address

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

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

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

Addressing

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

Examples:

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

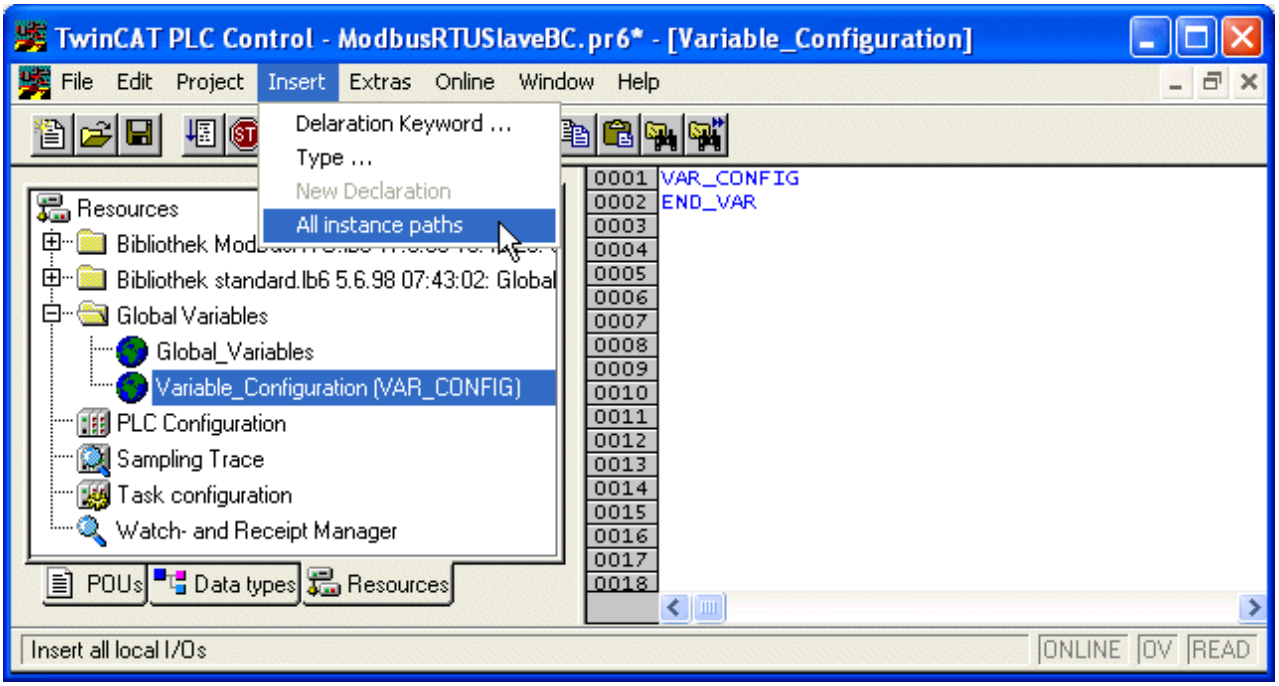
9 Modbus RTU Error Codes

```
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
```

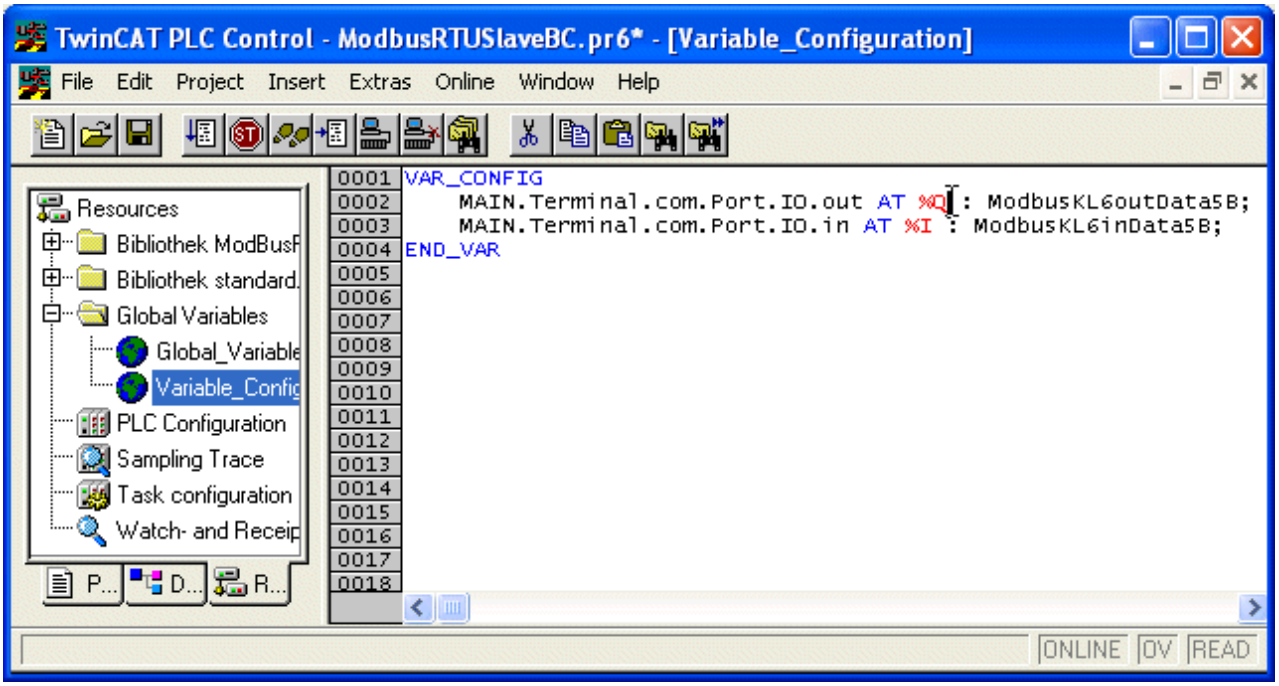
10 Hardware assignment at the BC bus controller

For a PC controller or a CX1000 controller, the serial port used is linked with the Modbus RTU function block within the TwinCAT System Manager. In contrast, in a program for a BC bus controller, this assignment is carried out manually directly in the PLC programming environment.

Once an instance of a Modbus RTU function block has been created in the PLC program, the hardware address of the serial Bus Terminal to be addressed is specified in the *variable configuration*. To this end, the menu item *Insert - All Instance Paths* is selected under *Resources* in the *Variable_Configuration (VAR_CONFIG)* block.



When this menu item is selected, the local I/O addresses used in all function blocks are listed.



Initially, the variables have no address. It has to be assigned manually according to the Bus Terminal assignment of the bus controller. If the Bus Terminal is the first non-digital terminal at the bus controller, the address is zero, for example.

```
VAR_CONFIG MAIN.Terminal.com.Port.IO.out AT %QB0 : ModbusKL6outData5B;
    MAIN.Terminal.com.Port.IO.in AT %IB0 : ModbusKL6inData5B;
END_VAR
```

11 Terminal configuration

The Bus Terminals KL6001, KL6011, KL6021, KL6031 and KL6041 can be parameterised 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

Development environment	Target system type	PLC libraries to be linked
TwinCAT from V2.8	PC (i386), CX1000	ComLibV2.lib or alternatively KL6config.lib
TwinCAT from V2.8	Bus controller BC	ComLibV2.lb6 or alternatively KL6config.lb6

More Information:
www.beckhoff.com/ts6255

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