

## **Setup of Modbus RTU- and serial communication in CODESYS | UR20-1COM-232-485-422-V2**

### **Abstract:**

The Application Note describes how to set up a Modbus RTU- and a serial communication with a u-remote UR20-1COM-232-485-422-V2 device in CODESYS. The examples explain the use of the UR20-1COM-232-485-422-V2 both via the System Bus of a UC20-WL2000, and via an EtherCAT u-remote fieldbus coupler.

### Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UC20-WL2000-AC	1334950000	u-OS 2.0.1
2	IoT-GW30	2682620000 2682630000	u-OS 2.0.1
3	UR20-1COM-232-485-422-V2	2826800000	FW 1.00.0
4	UR20-FBC-EC	1334910000	HW 2.04

### Software reference

No.	Software name	Article No.	Software version
1	CODESYS Development System		SP18 Patch 4
2	CODESYS Runtime App		4.7.0.0-2
3	CODESYS Control SL for Weidmueller u-OS		4.7.0.0

### Contact

Weidmüller Interface GmbH & Co. KG  
Klingenbergstraße 26  
32758 Detmold, Germany  
[www.weidmueller.com](http://www.weidmueller.com)

For any further support please contact your  
local sales representative:  
<https://www.weidmueller.com/countries>

Content

1 Warning and Disclaimer..... 4

2 Introduction..... 5

3 Install Modbus RTU and serial library package..... 6

4 FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusMaster ..... 7

5 FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusSlave ..... 8

6 FB\_UR20\_1ComRs\_232\_485\_422\_V2\_Serial..... 9

7 Create a Modbus RTU or serial communication ..... 10

7.1 Modbus RTU or serial communication via System\_Bus ..... 10

7.2 Modbus RTU or Serial Communication via u-remote coupler..... 11

7.3 Import and use of function block..... 13

7.4 Example Project..... 14

# 1 Warning and Disclaimer

## Warning

Controls may fail in unsafe operating conditions, causing uncontrolled operation of the controlled devices. Such hazardous events can result in death and / or serious injury and / or property damage. Therefore, there must be safety equipment provided / electrical safety design or other redundant safety features that are independent from the automation system.

## Disclaimer

This Application Note / Quick Start Guide / Example Program does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Each user is responsible for the correct operation of his control system. By using this Application Note / Quick Start Guide / Example Program prepared by Weidmüller, you accept that Weidmüller cannot be held liable for any damage to property and / or personal injury that may occur because of the use.

## Note

The given descriptions and examples do not represent any customer-specific solutions, they are simply intended to help for typical tasks. The user is responsible for the proper operation of the described products. Application notes / Quick Start Guides / Example Programs are not binding and do not claim to be complete in terms of configuration as well as any contingencies. By using this Application Note / Quick Start Guide / Example Program, you acknowledge that we cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this application note / quick start guide / example at any time without notice. In case of discrepancies between the proposals Application Notes / Quick Start Guides / Program Examples and other Weidmüller publications, like manuals, such contents have always more priority to the examples. We assume no liability for the information contained in this document. Our liability, for whatever legal reason, for damages caused using the examples, instructions, programs, project planning and performance data, etc. described in this Application Note / Quick Start Guide / Example is excluded.

## Security notes

In order to protect equipment, systems, machines and networks against cyber threats, it is necessary to implement (and maintain) a complete state-of-the-art industrial security concept. The customer is responsible for preventing unauthorized access to his equipment, systems, machines and networks. Systems, machines and components should only be connected to the corporate network or the Internet if necessary and appropriate safeguards (such as firewalls and network segmentation) have been taken.

## 2 Introduction

This application note describes the individual steps for implementing a Modbus RTU Master/Slave- and a serial communication between a Weidmüller u-OS PLC with a u-remote UR20-1COM-232-485-422-V2 module and a PC. The PC is used as example for any serial or modbus device and connects via a USB-to-RS485- or USB-to-RS232 adapter. It runs a simple serial terminal or a simulation program, emulating a Modbus RTU Master/slave device. We recommend using the free Modbus slave tool “ModRssim2”<sup>1</sup> or “QModMaster”<sup>2</sup> for a free Modbus Master simulator.

For the serial communication we recommend the free tool “hterm”<sup>3</sup> for Windows or “cutecom” for Linux.

---

<sup>1</sup> <https://sourceforge.net/projects/modrssi2/>

<sup>2</sup> <https://sourceforge.net/projects/qmodmaster/>

<sup>3</sup> <https://www.der-hammer.info/pages/terminal.html>

### 3 Install Modbus RTU and serial library package

1. Download the current CODESYS library files from the Weidmueller support center.  
[https://support.weidmueller.com/support-center/search/results?query=CODESYS&ac=library%26functionblock\\_id](https://support.weidmueller.com/support-center/search/results?query=CODESYS&ac=library%26functionblock_id)
2. Unzip the download and start the installation of the package via double click.
3. Click on the checkbox on the left bottom and start the installation.
4. Open a new standard CODESYS project and add the installed library to the project. In the device tree ⇒ Library Manager ⇒ Add Library... ⇒ libWiUr20ComRs\_232\_485\_422\_V2.

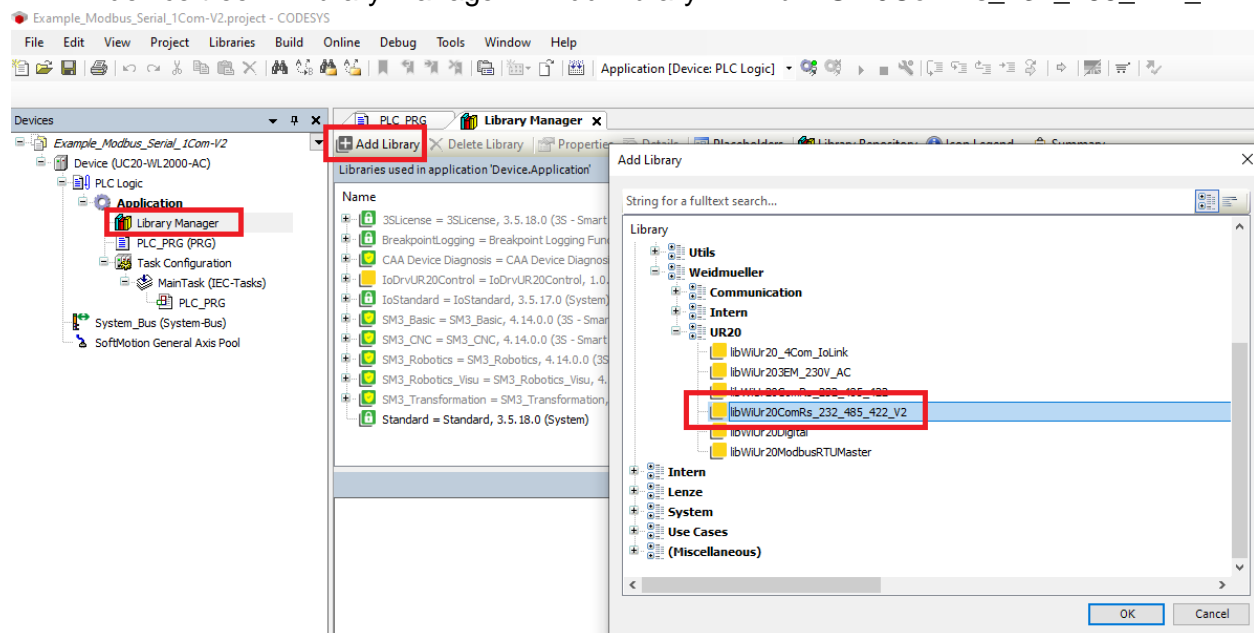


Figure 1: Add Library

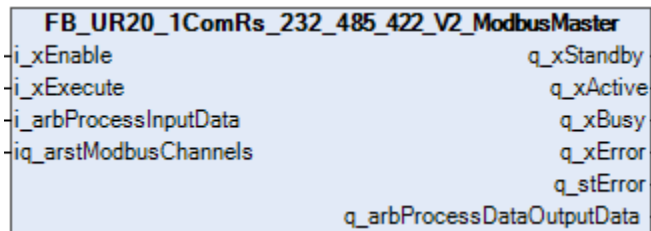


#### Note:

You can find the documentation of every function block in the downloaded zip file. Please read the documentation before use.

## 4 FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusMaster

The function block provides an interface to the UR20-1Com-RS-232-485-422-V2 module in its modbus master configuration. The module allows the user to parametrize up to 255 modbus master channels to exchange data.



**Figure 2: FB Modbus Master**

The behavior, functionality and the inputs/outputs of the function block are explained in the documentation of the function block.

Note that the documentation can also be found directly in the function block preview in the library manager.

The screenshot shows the CODESYS library manager with the following details:

- Library Structure:**
  - IOVUNL01\_001 = IOVUNL01\_001 (Weidmüller Interface GmbH & Co. KG)
    - IoStandard = IoStandard, 3.5.17.0 (System)
    - libWU20ComRs\_232\_485\_422\_V2 = libWU20ComRs\_232\_485\_422\_V2, 1.1.3.3 (Weidmüller)**
      - SM3\_Basic = SM3\_Basic, 4.14.0.0 (3S - Smart Software Solutions GmbH)
      - SM3\_CNC = SM3\_CNC, 4.14.0.0 (3S - Smart Software Solutions GmbH)
      - SM3\_Robotics = SM3\_Robotics, 4.14.0.0 (3S - Smart Software Solutions GmbH)
      - SM3\_Robotics\_Vis = SM3\_Robotics\_Vis, 4.13.0.0 (3S - Smart Software Solutions GmbH)
      - SM3\_Transformation = SM3\_Transformation, 4.14.0.0 (3S - Smart Software Solutions GmbH)
    - Standard = Standard, 3.5.18.0 (System)

- Details about selected library element 'FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusMaster'**
- FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusMaster (FB)**
- FUNCTION\_BLOCK FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusMaster EXTENDS FB\_LevelControlledBehavior
- Weidmüller**
- Functional Description**

The function block provides an interface to the UR20-1Com-RS-232-485-422-V2 module in its modbus master configuration. The module allows the user to parametrize up to 255 modbus master channels to exchange data. Every channel has its own message configuration with a function code, a data offset, a length and a cycle time. The function block transfers these parameters to the UR20-1Com-RS-232-485-422-V2 module in the standby initialize state and before the main action is started.

The channels are configured by the use of a variable array named `iq_arstModbusChannels`. To use it, the user needs to define its size. Every communication channel used, needs an array element. If for example the user wants to use three channels of the module, the size needs to be configured like `[0..2]`. The elements contain both, the channel configuration and the modbus register/coil data. After the configuration data is set, the user needs to enable the function block with the `i_xEnable` input. During this stage, the block resets all module data and writes the new configuration. A read data channel cyclically reads the data specified by variables "uiOffsetRead" and "uiLengthRead" in `iq_arstModbusChannels` and stores them in `arvRegisterData` or `arxCoilData` in `iq_arstModbusChannels`. A write data channel has two modes of operation: cyclic or on demand.

Cyclic: set `iq_arstModbusChannels[<channel number>].uiCycleTime` to the desired cycle time in milliseconds. The UR20\_1COM\_232\_485\_422\_V2 module will send the associated data every `iq_arstModbusChannels[<ch. no.>].uiCycleTime` milliseconds.

**Figure 3: function block documentation**

## 5 FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusSlave

This function block provides an interface to the UR20-1Com-RS-232-485-422-V2 module in its Modbus slave configuration. In this mode the module acts as a Modbus slave device.

The module allows the user to parametrize up to 255 Modbus slave channels to exchange data.

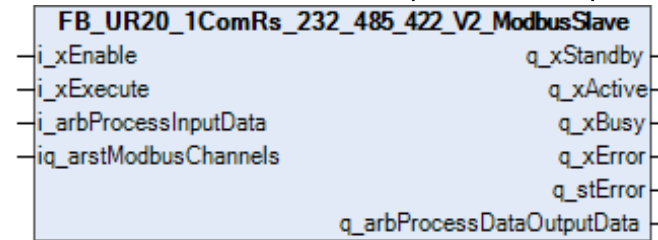


Figure 4: FB\_Modbus\_Slave

The behavior, functionality and the inputs/outputs of the function block are explained in the documentation of the function block.

Please find the information also directly in the function block preview in the library manager.

The screenshot shows the library manager with the selected library **libWUr20ComRs\_232\_485\_422\_V2**. The details pane shows the function block **FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusSlave (FB)**.

**FUNCTION\_BLOCK FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusSlave EXTENDS FB\_LevelControlledBehavior**

**Weidmüller**

**Functional Description**

This function block provides an interface to the UR20-1Com-RS-232-485-422-V2 module in its modbus slave configuration. In this mode the module acts as a Modbus slave device. The module allows the user to parametrize up to 255 modbus slave channels to exchange data. Each channel has its own data register which can be accessed via modbus commands. The data is stored in form of registers (16Bit) that can be accessed either as coils or registers.

The channels are configured in a variable-size array `iq_arstModbusChannels`. The user configures one array element per communication channel. E.g. for usage of three channels of the module, implement an array [0..2] and connect it to `iq_arstModbusChannels`. The array element associated to a channel contains both the configuration and the modbus register/coil data. The user first sets up the configuration data, then enables the function block by setting `i_xEnable` to true. The function block will then reset all module data and write the new configuration to the module.

After the function block has parametrized the desired communication channels, the function block confirms success by setting output `q_xStandby` to true. Next it waits for the execute signal `i_xExecute` before it enters the state active and starts its main operation. While the block is active, the channel configuration cannot be changed.

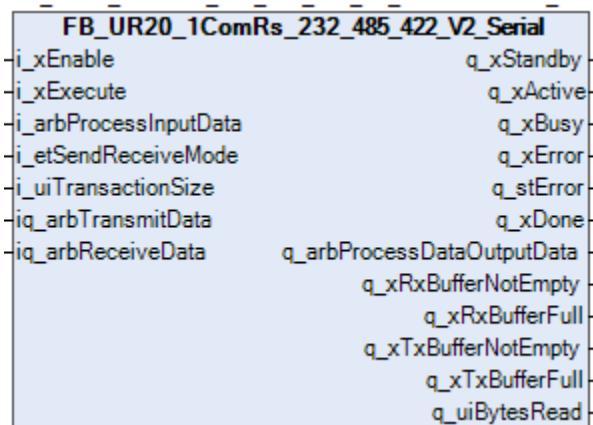
During the startup of the main operation the function block initially writes the register data in `iq_arstModbusChannels` to `wRegisters` to the module. In ongoing active state, the function block cyclicly checks the state of `xTriggerTransaction` for each member of `iq_arstModbusChannels`. The function block exchanges data with the 1COM V2 module when `xTriggerTransaction` of a channel is true. The transfer direction depends on `iq_arstModbusChannels[channel number].xTransactionRead`. If `iq_arstModbusChannels[channel number].xTransactionRead` is true, the function block will read channel data from the 1COM V2 module and store them in `iq_arstModbusChannels[channel number].wRegisters`. If `iq_arstModbusChannels[channel number].xTransactionRead` is false, the function block will write channel data from `iq_arstModbusChannels[channel number].wRegisters` into the 1COM V2 module. After the data transfer is

Figure 5: FB Modbus slave documentation



## 6 FB\_UR20\_1ComRs\_232\_485\_422\_V2\_Serial

The function block provides an interface to the UR20-1Com-RS-232-485-422-V2 module in the custom mode configuration. In the custom mode it is possible to create and adjust a serial communication between the module and another serial device connected via RS232, RS422 or RS485.



**Figure 6: FB for serial communication**

The behavior, functionality and the inputs / outputs of the function block are explained in the documentation of the function block.

This documentation also is directly available in the library repository when you open the function block.

The video [Introduction to u-remote function block libraries for Codesys - YouTube](#) is also a good source of information. The video explains how to install a library and how to use the function blocks.

## 7 Create a Modbus RTU or serial communication

This chapter explains how to create a Modbus RTU or a serial communication.

It is essential to know the parameters and settings of the connected devices. You need to set up the communication parameters of the UR20-1COM-232-485-422-V2 accordingly. In the example we use the following parameters:

### Example:

Operating mode:	RS485
Baud rate:	9600 kbps
Parity:	none
Stop bits:	1
Flow control:	none
Data bits:	8 bit
Terminating resistor:	On

### 7.1 Modbus RTU or serial communication via System\_Bus

1. In the Device Tree, open the System\_Bus configurator either by double-clicking the *System\_Bus* or by right-click on the *System\_Bus* and then choosing the menu item *Add Device*. Add the UR20\_1COM\_232\_485\_422\_V2 module to the System\_Bus.
2. Double-click the UR20\_1COM\_232\_485\_422\_V2, open *Module Configuration* and adjust its communication parameters.

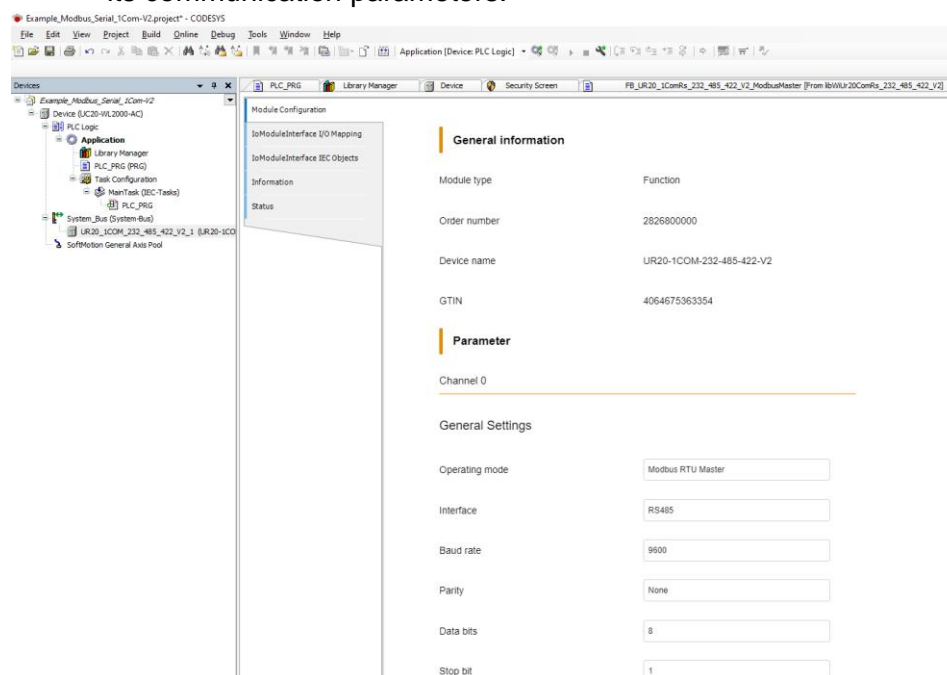


Figure 7: Communication Parameter

3. Map variables to the IO of the UR20\_1COM\_232\_485\_422\_V2. Therefore, select *IoModuleInterface I/O Mapping* in the module representation. Then map one variable after the

other to the channels: click into the variable field and select the variable you want to associate with this channel. See Figure 8: Variable <-> I/O Channel mapping.



### Note:

Theoretically, the CODESYS ST syntax allows to associate variables to I/O channels by assigning an absolute address to a variable, e.g.:

```
i_arusiMBInput AT%IB0 : ARRAY[0..18] OF USINT;
```

However, this is not robust against changes of the station setup. CODESYS has declared this syntax deprecated and Weidmüller advises not to use it.

Variable	Mapping	Channel	Address	Type	Current Value
Application.GVL_IO.i_arusiMBInput[0]		Input	%IB22	BYTE	144
Application.GVL_IO.i_arusiMBInput[1]		Input	%IB23	BYTE	0
Application.GVL_IO.i_arusiMBInput[2]		Input	%IB24	BYTE	144
Application.GVL_IO.i_arusiMBInput[3]		Received data 1	%IB25	USINT	0
Application.GVL_IO.i_arusiMBInput[4]		Received data 2	%IB26	USINT	16
Application.GVL_IO.i_arusiMBInput[5]		Received data 3	%IB27	USINT	0
Application.GVL_IO.i_arusiMBInput[6]		Received data 4	%IB28	USINT	77
Application.GVL_IO.i_arusiMBInput[7]		Received data 5	%IB29	USINT	0
Application.GVL_IO.i_arusiMBInput[8]		Received data 6	%IB30	USINT	88
Application.GVL_IO.i_arusiMBInput[9]		Received data 7	%IB31	USINT	12
Application.GVL_IO.i_arusiMBInput[10]		Received data 8	%IB32	USINT	128
Application.GVL_IO.i_arusiMBInput[11]		Received data 9	%IB33	USINT	0
Application.GVL_IO.i_arusiMBInput[12]		Received data 10	%IB34	USINT	0
Application.GVL_IO.i_arusiMBInput[13]		Received data 11	%IB35	USINT	0
Application.GVL_IO.i_arusiMBInput[14]		Received data 12	%IB36	USINT	0
Application.GVL_IO.i_arusiMBInput[15]		Received data 13	%IB37	USINT	0
Application.GVL_IO.i_arusiMBInput[16]		Received data 14	%IB38	USINT	0
Application.GVL_IO.i_arusiMBInput[17]		Received data 15	%IB39	USINT	0
Application.GVL_IO.i_arusiMBInput[18]		Received data 16	%IB40	USINT	0
Application.GVL_IO.q_arusiMBOutput[0]		Output	%QB21	BYTE	16
Application.GVL_IO.q_arusiMBOutput[1]		reserved	%QB22	USINT	0
Application.GVL_IO.q_arusiMBOutput[2]		Output	%QB23	BYTE	1
Application.GVL_IO.q_arusiMBOutput[3]		Transmission data 1	%QB24	USINT	0
Application.GVL_IO.q_arusiMBOutput[4]		Transmission data 2	%QB25	USINT	0
Application.GVL_IO.q_arusiMBOutput[5]		Transmission data 3	%QB26	USINT	0
Application.GVL_IO.q_arusiMBOutput[6]		Transmission data 4	%QB27	USINT	10
Application.GVL_IO.q_arusiMBOutput[7]		Transmission data 5	%QB28	USINT	0
Application.GVL_IO.q_arusiMBOutput[8]		Transmission data 6	%QB29	USINT	20
Application.GVL_IO.q_arusiMBOutput[9]		Transmission data 7	%QB30	USINT	0
Application.GVL_IO.q_arusiMBOutput[10]		Transmission data 8	%QB31	USINT	55
Application.GVL_IO.q_arusiMBOutput[11]		Transmission data 9	%QB32	USINT	0
Application.GVL_IO.q_arusiMBOutput[12]		Transmission data 10	%QB33	USINT	99
Application.GVL_IO.q_arusiMBOutput[13]		Transmission data 11	%QB34	USINT	0
Application.GVL_IO.q_arusiMBOutput[14]		Transmission data 12	%QB35	USINT	0
Application.GVL_IO.q_arusiMBOutput[15]		Transmission data 13	%QB36	USINT	0
Application.GVL_IO.q_arusiMBOutput[16]		Transmission data 14	%QB37	USINT	0
Application.GVL_IO.q_arusiMBOutput[17]		Transmission data 15	%QB38	USINT	0
Application.GVL_IO.q_arusiMBOutput[18]		Transmission data 16	%QB39	USINT	0

Figure 8: Variable <-> I/O Channel mapping

## 7.2 Modbus RTU or Serial Communication via u-remote coupler



### Note:

Read the u-remote manual for information on how to set up the parameter values of the UR20\_1COM\_232\_485\_422\_V2 via an u-remote fieldbus coupler. Here is a download link to the [u-remote manual](#).

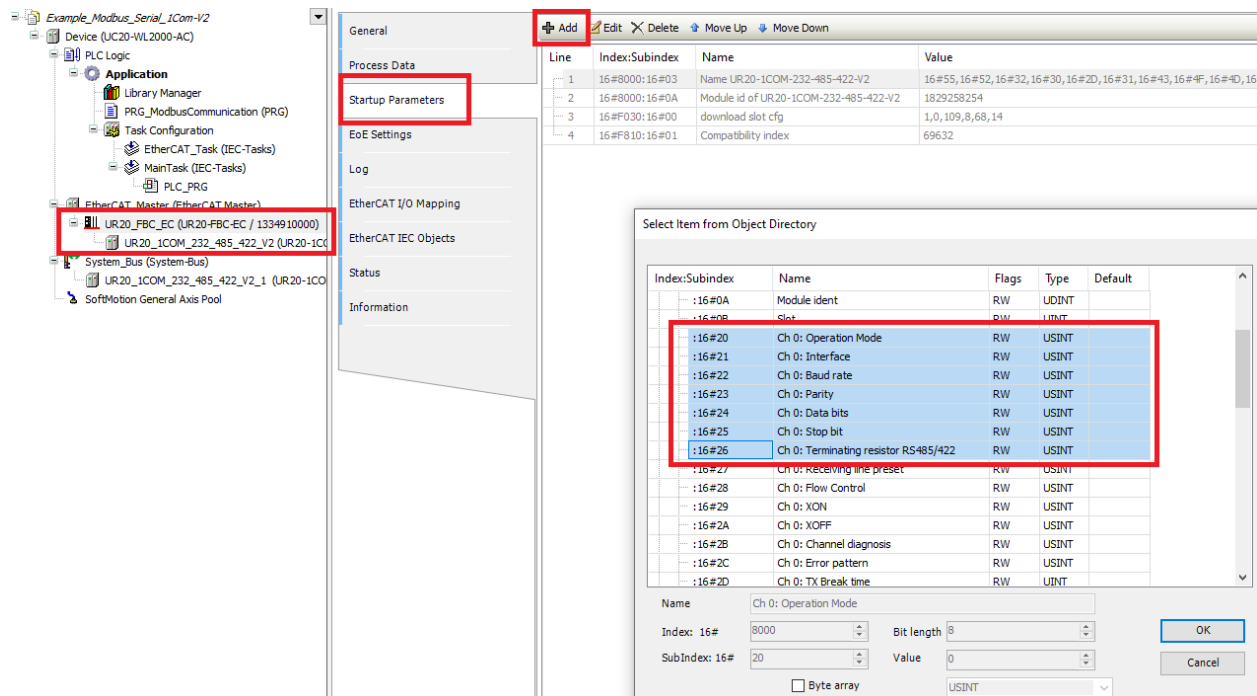


### Note:

This example requires the CODESYS EtherCAT package and the device description files for the Weidmüller u-remote EtherCAT fieldbus coupler. Please install the EtherCAT package via the CODESYS Installer. Download and install the EtherCAT device description files from Weidmüller's [product homepage](#). => Downloads => Software => current ESI file for you HW version.

In this example we use an EtherCAT fieldbus coupler to connect a UR20\_1COM\_232\_485\_422\_V2 to our PLC.

1. In the Device Tree of your CODESYS project, take the following steps:
  - a. Add an EtherCAT Master to your project.
  - b. In the EtherCAT master => General => EtherCAT NIC settings, click "Select..." to the right of the Source address (MAC). Assuming you have connected X2 of your UC20-WL2000-AC with the EtherCAT coupler, select eth1.
  - c. Scan the EtherCAT fieldbus for the connected devices. Right-click on EtherCAT\_Master => Scan for devices.. => copy to project
  - d. Set up the communication parameters in the *Startup Parameters* of the EtherCAT coupler:
    - i. open UR20\_FBC\_EC => Startup Parameters => Add => Parameter UR20\_1COM\_232\_485\_422\_V2. See Figure 9: Add Parameter.
    - ii. Select the parameters to adjust and click ok.
    - iii. Edit the values for the desired communication.



**Figure 9: Add Parameter**

Please find an overview of module parameters in the u-remote manual. Figure 10: Module Parameter shows a part of these for the UR20\_1COM-232\_485\_422\_V2 module.

Overview of the editable parameter UR20-1COM-232-485-422-V2

Description	Options	Default
<b>General settings</b>		
Operating mode	disabled (0) / custom (1) / Modbus RTU Master (2) / Modbus ASCII Master (3) / Modbus RTU Slave (4) / Modbus ASCII Slave (5) / DMX Master (6) / DMX Slave (7)	disabled
Interface	RS232 (0) / RS485 (1) / RS422 (2)	RS232
Baud rate	300 (0) / 600 (1) / 1200 (2) / 1800 (3) / 2400 (4) / 4800 (5) / 9600 (6) / 14400 (7) / 19200 (8) / 28800 (9) / 38400 (10) / 57600 (11) / 115200 (12) / 230400 (13)	9600
Parity	None (0) / Odd (1) / Even (2) / Space (3) / Mark (4) / Any (5)	None
Data bits	8 Bits (0) / 7 Bits (1)	8 Bits
Stop bit	1 Bit (0) / 1.5 Bits (1) / 2 Bits (2)	1 Bit
Terminating resistor RS485/422	disabled (0) / enabled (1)	disabled
Receiving line preset	disabled (0) / Idle termination (1) / Line break termination (2)	disabled
<b>Flow control</b>		
Flow control	disabled (0) / CTS/RTS (1) / XON/XOFF (2)	disabled
XON	0 ... 255	17 (0x11)
XOFF	0 ... 255	19 (0x13)
<b>Error handling</b>		
Channel diagnosis	disabled (0) / enabled (1)	enabled
Error pattern	0 ... 255 <sup>1)</sup>	255
<b>TX settings</b>		
Break time	0 ... 65535 Bits	12
Idle time	0 ... 65535 Bits	40
Send break before start telegram	Off (0) / On (1)	Off

Figure 10: Module Parameter

The process data is mapped exactly in the same way as when using the module via the System Bus.

### 7.3 Import and use of function block

1. Add the library "libWiUr20ComRs\_232\_485\_422\_V2" to your PLC project.
2. Create a new program and use the function block needed in your application: for Modbus RTU Master communication "FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusMaster", for Modbus slave communication "FB\_UR20\_1ComRs\_232\_485\_422\_V2\_ModbusSlave" and for serial communication "FB\_UR20\_1ComRs\_232\_485\_422\_V2\_Serial".
3. Create a program sequence to execute the function block.

In addition to this application note, Weidmüller provides a CODESYS example project that demonstrates usage of the relevant function blocks in libWiUr20ComRs\_232\_485\_422\_V2. The next chapter introduces this example project.

## 7.4 Example Project

The example project offers six programs for the following use cases, see Figure 11: Example Project. In the example shown, the program for Modbus RTU Master via System\_Bus is activated.

- Modbus RTU Master via EtherCAT in PRG\_ModbusMasterCommunication\_Ethercat
- Modbus RTU Slave via EtherCAT in PRG\_ModbusSlaveCommunication\_Ethercat
- Serial communication via EtherCAT in PRG\_SerialCommunication\_Ethercat
- Modbus RTU via System\_Bus in PRG\_ModbusMasterCommunication\_Sysbus
- Modbus RTU via System\_Bus in PRG\_ModbusSlaveCommunication\_Sysbus
- Serial communication via System\_Bus in PRG\_SerialCommunication\_Sysbus

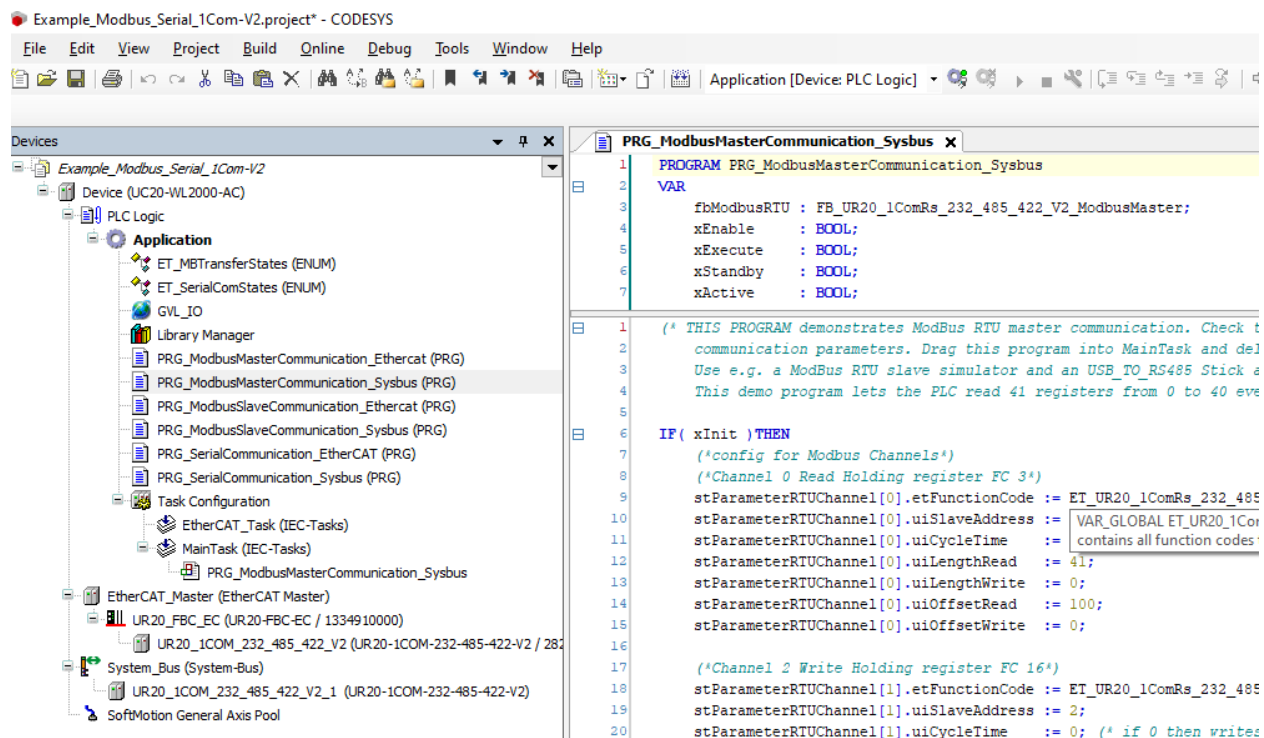


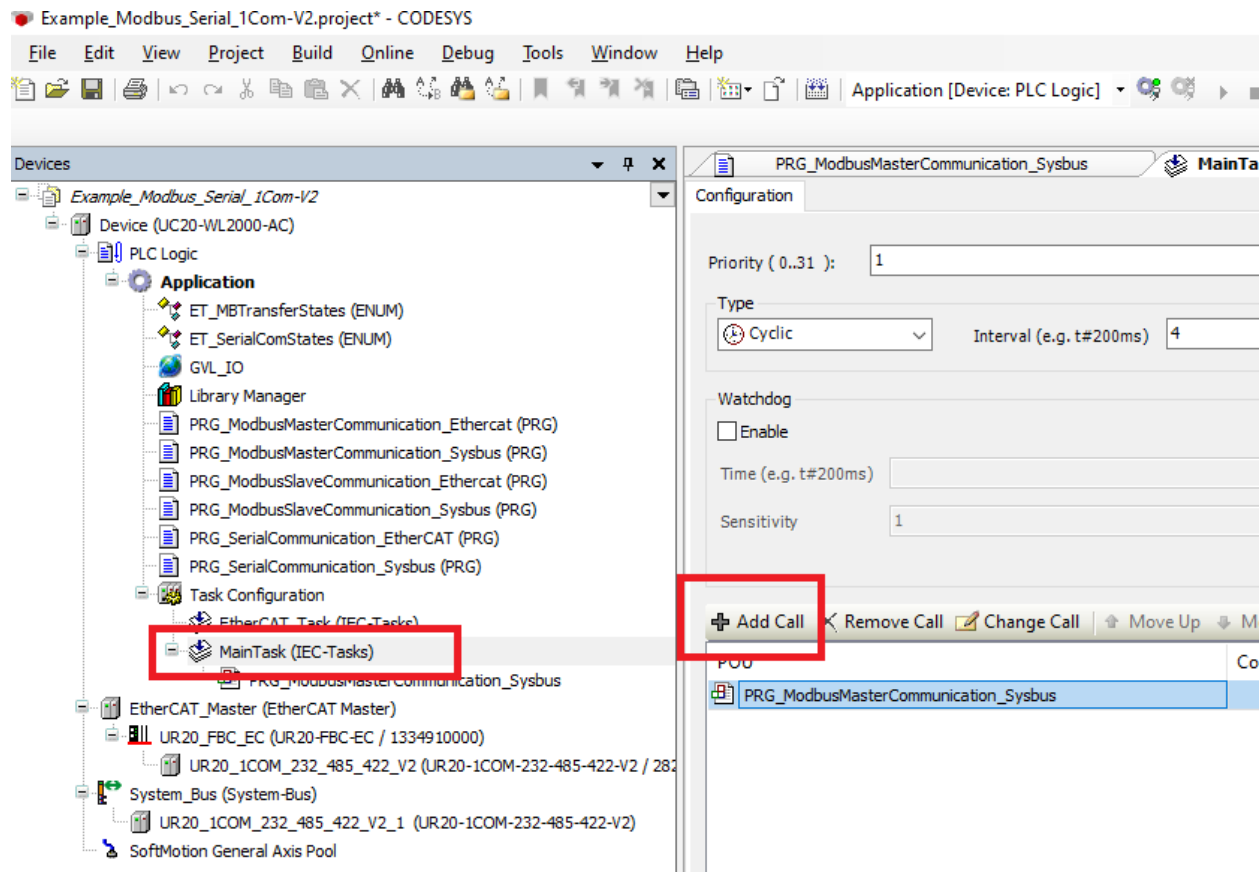
Figure 11: Example Project

To activate the desired program for your use case, add it to the MainTask: In the device tree ⇒ open Task Configuration ⇒ MainTask ⇒ Add Call ⇒ Select your program ⇒ ok. Similarly, you find “Remove Call” to exclude a program from execution. See Figure 12: Add or Remove Call.



### Note:

Multiple programs must not access the same module simultaneously because they overwrite each other's module I/O data. Therefore, in the example project, you can use one program only for the system bus and one program only for EtherCAT at the same time.



**Figure 12: Add or Remove Call**

### Modbus RTU Master examples:

These examples demonstrate how to run a Modbus RTU master in a UC20\_WL2000\_AC u-OS CODESYS application.

The demo programs are identical, with just one difference: The variables mapping the fbModBusRTU process image to the UR20\_1COM\_RS232\_485\_422\_V2 module depend on which field bus the example uses. Just drag the desired demo program into the MainTask and remove the other from the MainTask. See Figure 12: Add or Remove Call.

Configure the operation mode and communication settings in the parametrization of the UR20\_1COM\_232\_485\_422\_V2 module. Download the PLC program and login.

The Modbus RTU Master examples require a Modbus RTU slave as communication counterpart. We suggest that you use an USB<->485- or USB<->RS232 adapter and a Modbus simulator on your PC, e.g. ModRSSim2 for Windows.

Open the ModRSSim2 simulator and write some values to the registers. Start the PLC application. The Modbus RTU Master demo program has two channels configured for the communication to the slave device.

The first channel 0 reads some holding register (FC 3) from a slave device every 1000ms. It stores them in the array arOutputDataRegister.



Channel 1 writes holding registers (FC16). To write data, the trigger variable xTriggerTransaction of the channel has to be set to **true**. The write data must be set inside the array q\_arOutDataWord.

Change values in ModRSSim2 and CODESYS and observe the changes both in arOutputDataRegister in the CODESYS demo program and in the slave simulator.

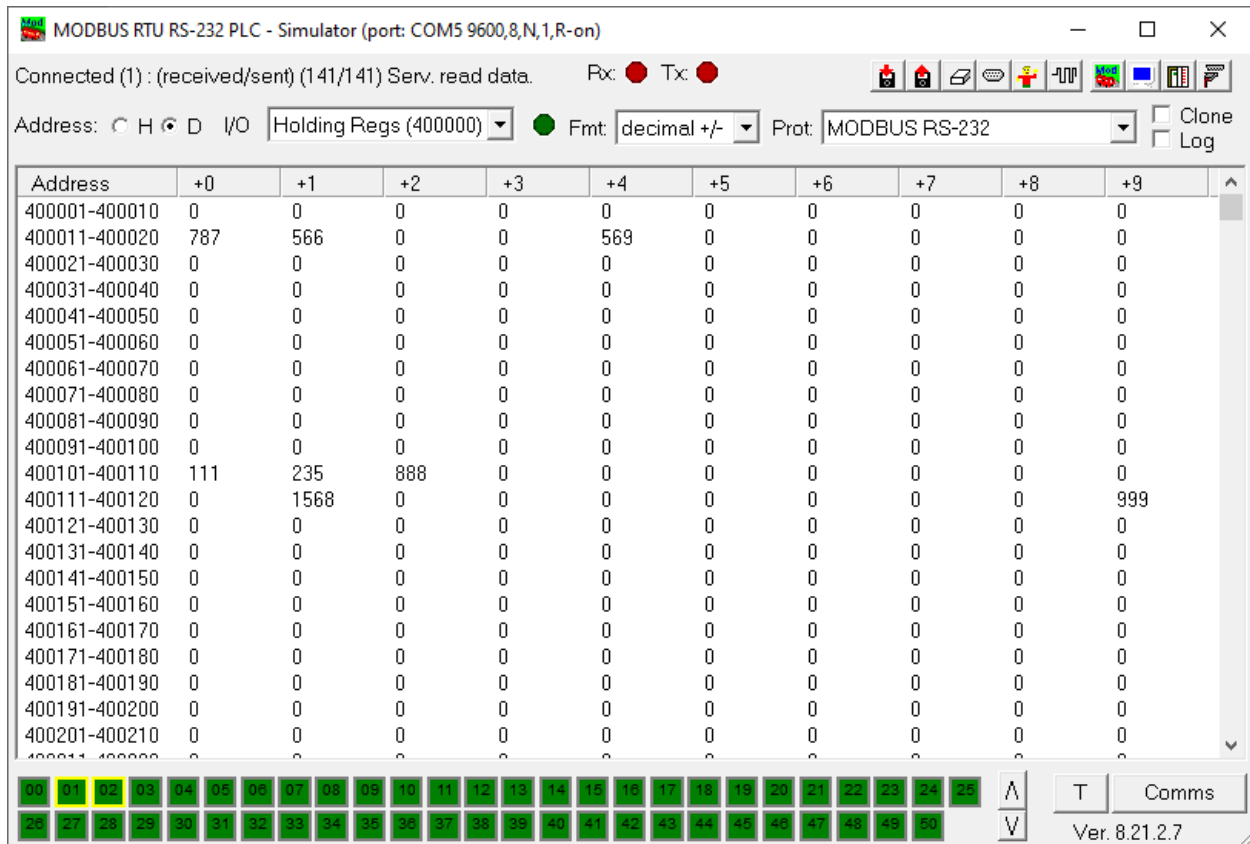


Figure 13: Modbus Simulator



## Setup of Modbus RTU- and serial communication in CODESYS | UR20-1COM-232-485-422-V2

Expression	Type	Value
arOutputDataRegister	ARRAY [0..124] OF ...	
arOutputDataRegister[0]	WORD	111
arOutputDataRegister[1]	WORD	235
arOutputDataRegister[2]	WORD	888
arOutputDataRegister[3]	WORD	0
arOutputDataRegister[4]	WORD	0
arOutputDataRegister[5]	WORD	0
arOutputDataRegister[6]	WORD	0
arOutputDataRegister[7]	WORD	0
arOutputDataRegister[8]	WORD	0
arOutputDataRegister[9]	WORD	0
arOutputDataRegister[10]	WORD	0
arOutputDataRegister[11]	WORD	1568
arOutputDataRegister[12]	WORD	0
arOutputDataRegister[13]	WORD	0
arOutputDataRegister[14]	WORD	0
arOutputDataRegister[15]	WORD	0
arOutputDataRegister[16]	WORD	0
arOutputDataRegister[17]	WORD	0
arOutputDataRegister[18]	WORD	0
arOutputDataRegister[19]	WORD	999

Figure 14: Read Modbus RTU values via System\_Bus

Expression	Type	Value
q_arOutDataCoil	ARRAY [1..250] OF ...	
q_arOutDataBool	ARRAY [1..2000] OF ...	
xStart	BOOL	TRUE
mbDoInit	BOOL	TRUE
etMainState	ET_MBTRANSFERST...	ModbusTransfer
arOutputDataRegister	ARRAY [0..124] OF ...	
arOutputDataRegister[0]	WORD	111
arOutputDataRegister[1]	WORD	235
arOutputDataRegister[2]	WORD	888
arOutputDataRegister[3]	WORD	0
arOutputDataRegister[4]	WORD	0
arOutputDataRegister[5]	WORD	0
arOutputDataRegister[6]	WORD	0
arOutputDataRegister[7]	WORD	0
arOutputDataRegister[8]	WORD	0
arOutputDataRegister[9]	WORD	0
arOutputDataRegister[10]	WORD	0
arOutputDataRegister[11]	WORD	1568
arOutputDataRegister[12]	WORD	0
arOutputDataRegister[13]	WORD	0

Figure 15: Read values via EtherCAT

### Serial communication:

The programs PRG\_SerialCommunication\_EtherCAT and PRG\_SerialCommunication\_Sysbus demonstrate serial communication. They, too, are nearly identical copies of each other. Just the mapping of the I/O data of fbSerial1 is different, either EtherCAT or the System\_Bus is selected.

The PRG\_SerialCommunication\_<Fieldbus> implements a simple state machine with the state variable `etMain` : `ET_SerialComStates`. This state machine initializes the `fbSerial` and puts it into its state **Standby**. Then it waits for incoming serial messages or for the user to trigger a “send serial data” sequence. After it has finished receiving or sending serial data, it enters the state `ET_SerialComStates.FINISHED`.

1. Select the PRG\_SerialCommunication\_<Fieldbus> of your choice.
2. Remove any other PRGs from the MainTask.
3. Download and login to the PLC.
4. Weidmüller recommends connecting the UR20\_1COM\_232\_485\_422\_V2 to a USB to serial converter and run e.g. `hterm` on a Windows PC or `CuteCom` on a Linux PC to view and input the incoming and outgoing serial data.
5. Start the application.
6. Set `xStart` to **true**.

The program now waits in its state `ET_SerialComStates.FB_IN_STANDBY` until data is available.

7. Send some serial data, e.g. “Hello!” via the terminal program of your choice.  
The serial communication FB signals available received data and the program enters the state `ET_SerialComStates.FB_ACTIVATED`. It transfers the received data to the variables `arReceiveDataSave` and `sTextOutput`. Then it jumps back to `ET_SerialComStates.FB_IN_STANDBY` and waits for a new telegram.

8. Observe the received data in arReceiveDataSave and sTextOutput.

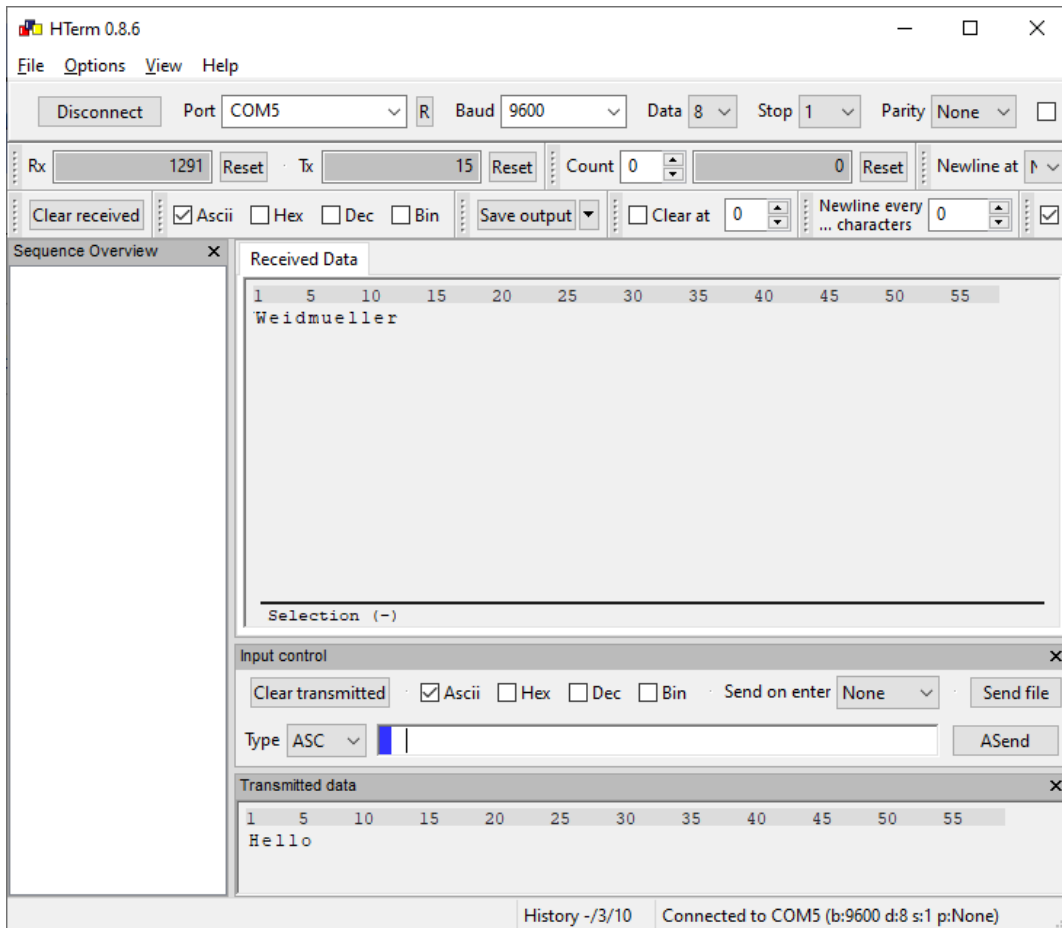
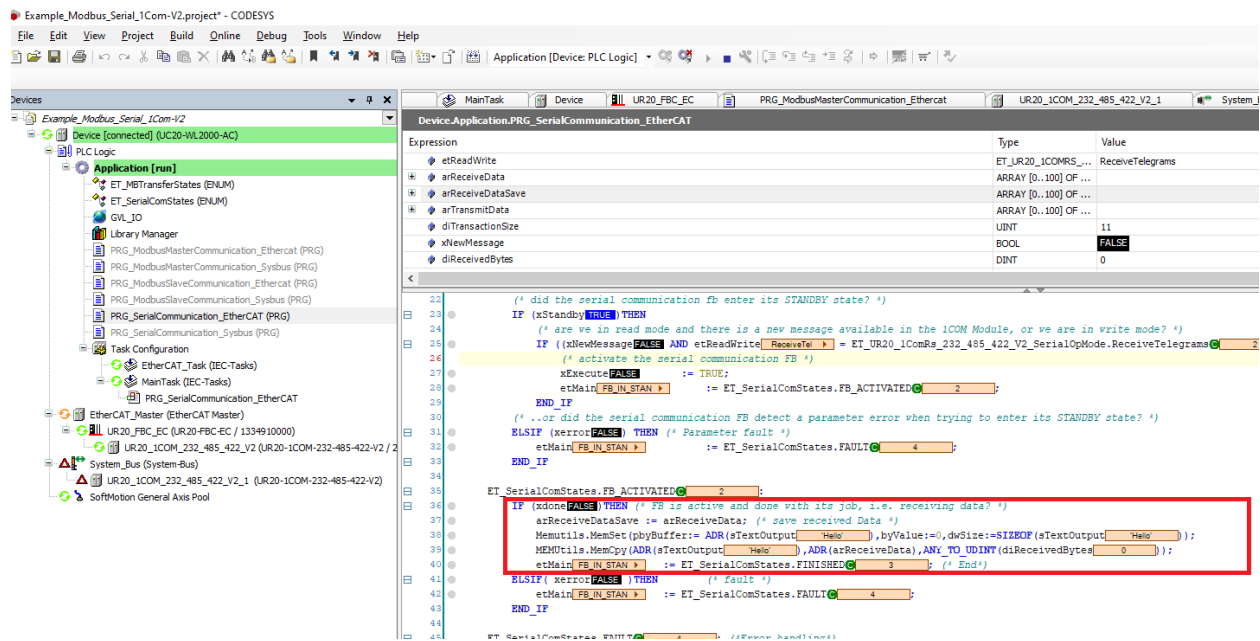


Figure 16: Send and receive serial data via hterm



**Figure 17: Receive Data**

## Modbus RTU Slave examples:

These examples demonstrate how to run a Modbus RTU slave in a UC20\_WL2000\_AC u-OS CODESYS application.

The demo programs for Sysbus and EtherCAT are identical with just one difference: the variables mapping the fbModBusRTUSlave process image to the UR20\_1COM\_RS232\_485\_422\_V2 module depend on which fieldbus the example uses. Just drag the desired demo program into the MainTask and remove the other from the MainTask, see Figure 12: Add or Remove Call.

Configure the operation mode and communication settings in the parameterization of the UR20\_1COM\_232\_485\_422\_V2 module. Download the PLC program and login.

The Modbus RTU slave examples require a Modbus RTU Master as communication counterpart. For testing, Weidmüller suggests that you use a USB-to-485- or USB-to-RS232 adapter and a Modbus Master simulator on your PC, e.g. QModMaster for Windows.

Start the PLC application. The Modbus RTU slave demo program has two channels configured for the communication to the master device.

The first channel 0 offers 100 holding register (FC 3) with start address 0. It is possible to read and to write data from/to the modbus slave register.

Channel 1 offers input registers (FC4). It is only possible to read data from the register.

To write data from PLC to the Modbus Register on the module, the trigger variable xTriggerTransaction of the channel must be set to **true**. The write data must be set inside the array arData1 or arData2. To read the current Modbus register values from the module to the PLC, the variable xTransactionRead must be set to **true** and the variable xTriggerTransaction must be triggered.

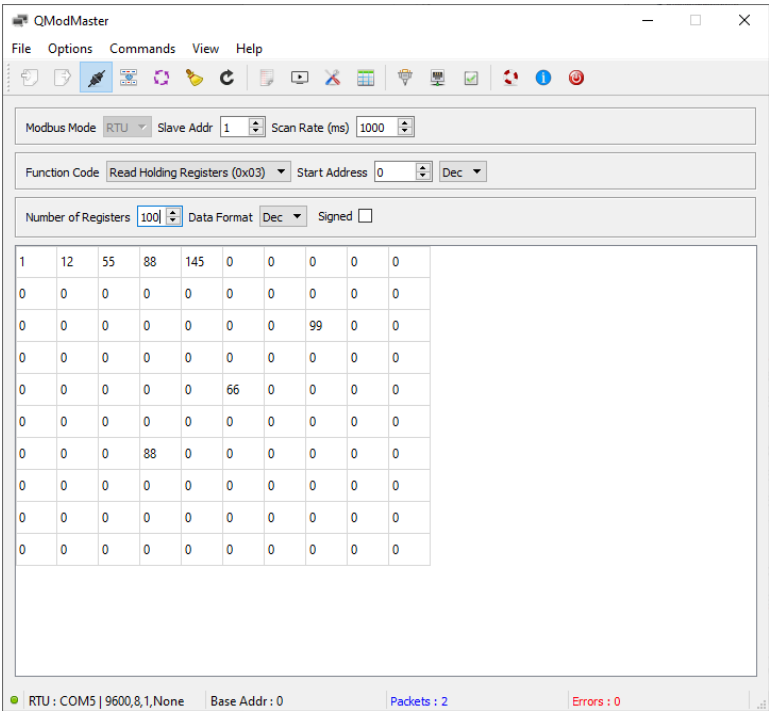


Figure 18: QModMaster Read data

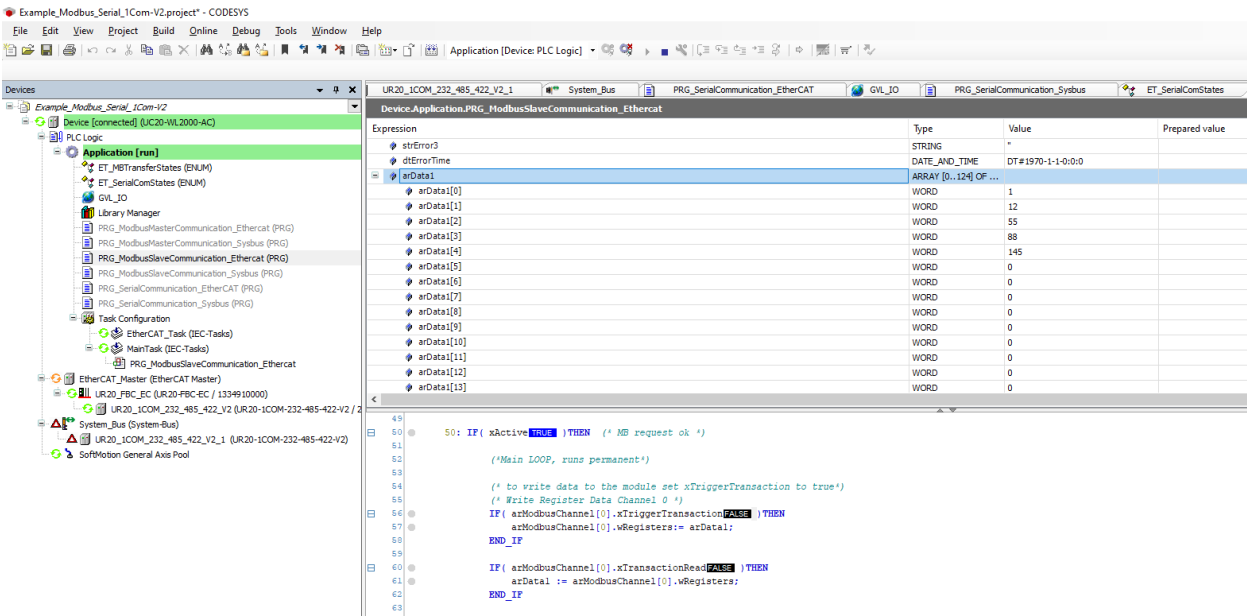
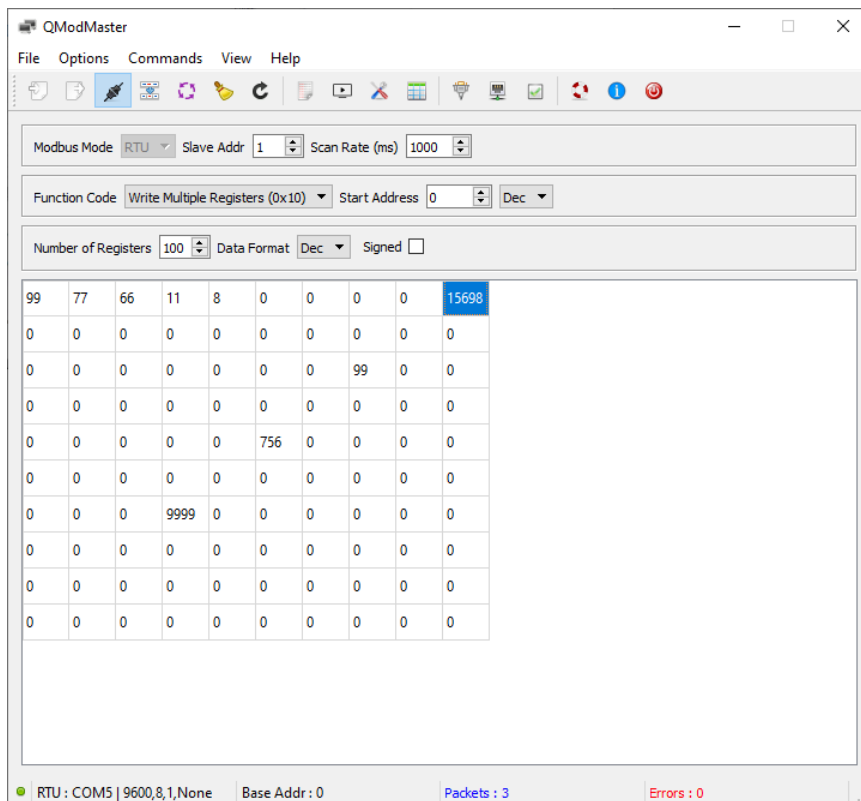
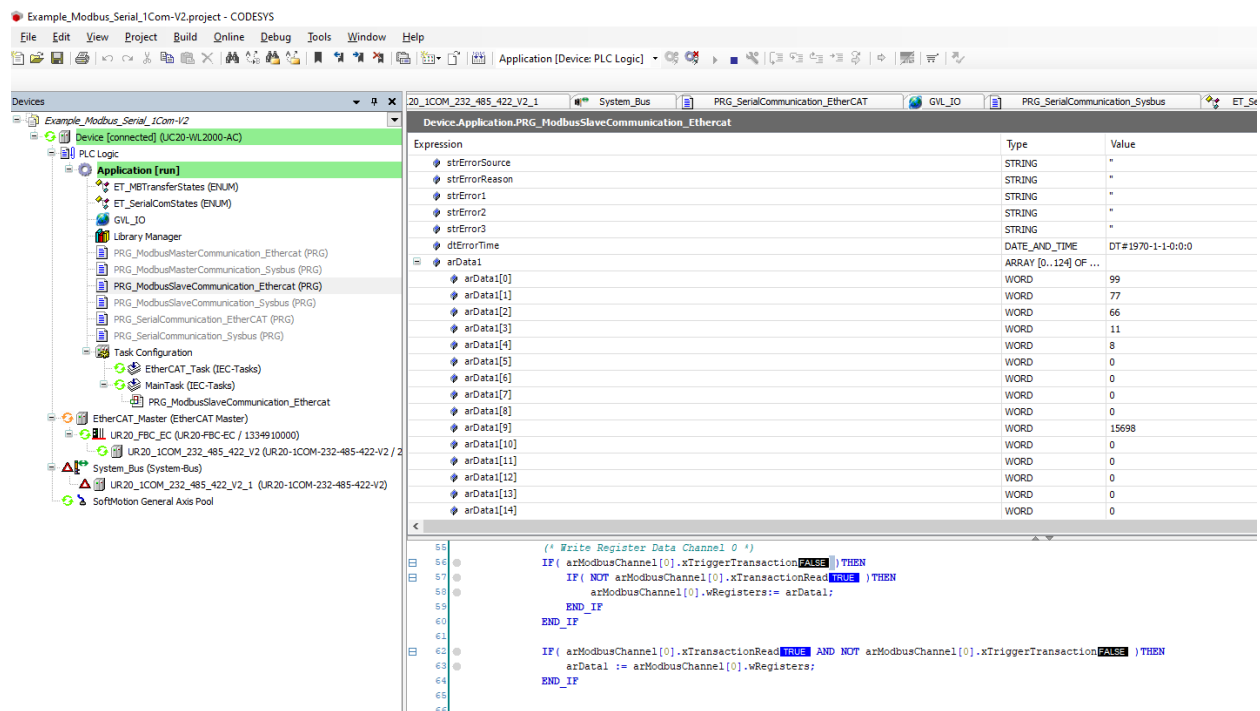


Figure 19: Modbus slave data register data



**Figure 20: QModMaster Write Data**



**Figure 21: Read data from module**