

UC20-SL2000-OLAC-EC

Application Note | Modbus TCP Client/Master

Abstract:

The Application Note describes how to set up a Modbus TCP communication on a UC20-SL2000-OLAC-EC controller. The document contains instructions how to integrate the Weidmüller Modbus TCP Client/Master library “libWIModbusTCPClient.library” into u-create studio software and how to use the Modbus TCP Client Function Block “FB_ModbusTCPClient” containing in the library to create a Modbus TCP communication.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UC20-SL2000-OLAC-EC	2638920000	-

Software reference

No.	Software name	Article No.	Software version
1	u-create studio	2660130000	1.20.2 (or higher)
2	ModRSSim2	-	-
3	libWiPackageStudio	-	1.0.0 (or higher)

File reference

No.	Name	Description	Version
1	AN0052v4-UC20-SL2000 ModbusTCPClient.zip	The file contains a Modbus TCP example project related to this document	-

Contact

Weidmüller Interface GmbH & Co. KG
Klingenbergstraße 26
32758 Detmold, Germany
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	Requirements	5
3	Install libWiPackageStudio.....	6
4	FB_ModbusTCPClient	7
4.1	i_xEnable (BOOL)	7
4.2	i_xExecute (BOOL).....	7
4.3	i_stParamInputs (STRUCT).....	7
4.4	Inputs	7
4.5	q_xStandby (BOOL)	8
4.6	q_xActive (BOOL).....	8
4.7	q_xBusy (BOOL)	8
4.8	q_xDone (BOOL).....	8
4.9	q_xError (BOOL)	8
4.10	q_stError (STRUCT).....	8
5	Create a Modbus TCP communication	9
5.1	Example Program.....	9
5.2	Run Example	11

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 Requirements

To work through this document, it is necessary to install the u-create studio software on an engineering PC and connect the patch cable between the controller and the PC.

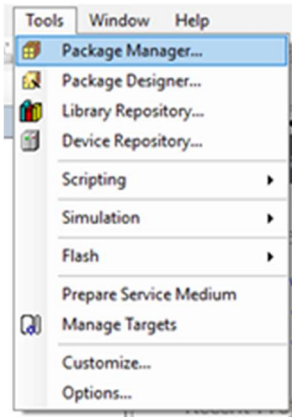
Furthermore, a Modbus TCP server simulator is used in the application example, which must also be executed on the engineering PC.



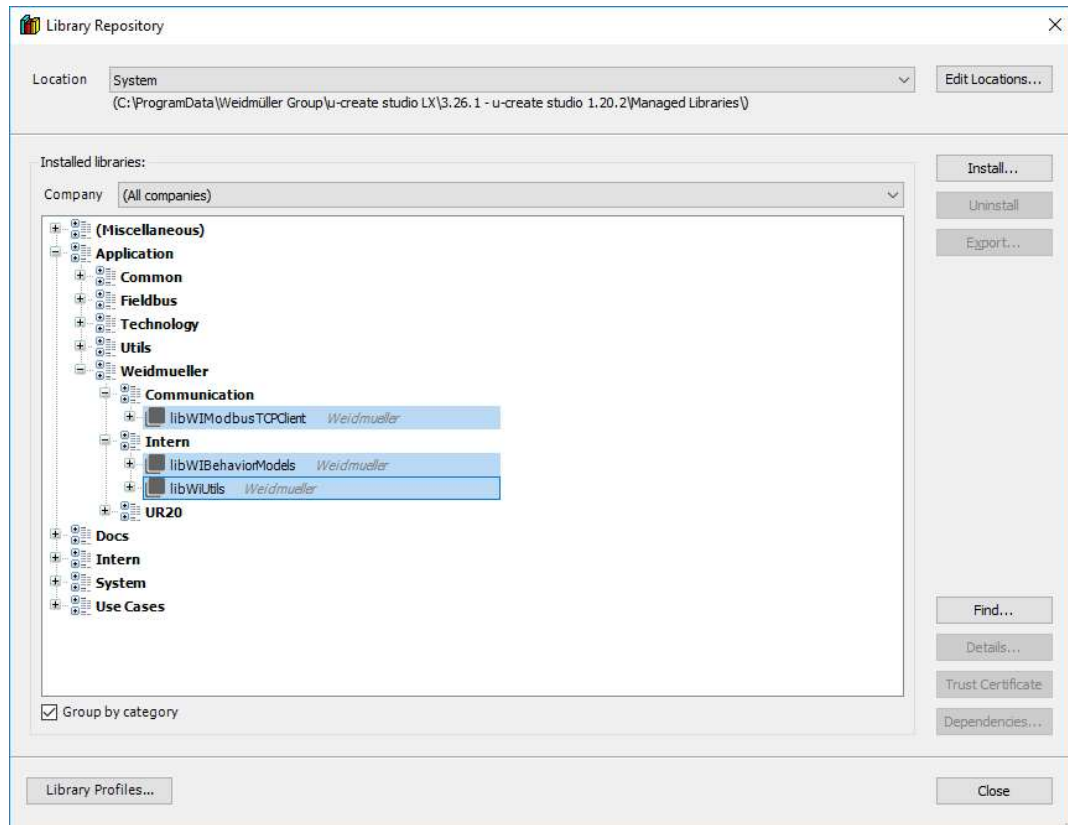
To understand the content of this document, basic knowledge about handling projects in u-create studio is required.

3 Install libWiPackageStudio

1. Open u-create studio and start the **Package Manager**.



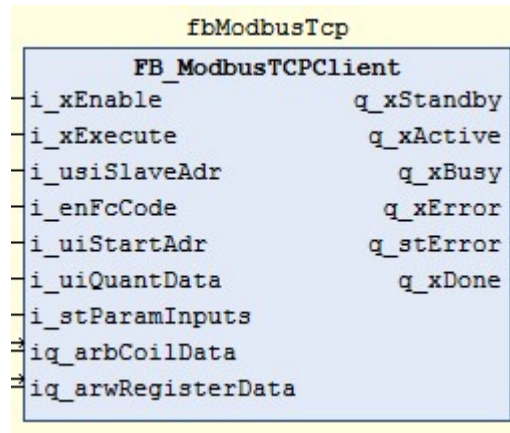
2. Install the package¹
3. Check the Library Repository if all necessary libraries have been installed.
 - libWiBehaviorModels
 - libWiModbusTCPClient
 - libWiUtils



¹ Download link: [libWiStudioPackage](#)

4 FB_ModbusTCPClient

The function block implements a Modbus TCP client that provides communication with a Modbus TCP server.



4.1 i_xEnable (BOOL)

Release of the function block when changing to a positive level. After a first check the module is ready for use.

4.2 i_xExecute (BOOL)

Starts the processing of the block when changing to a positive level. If the process values are valid, the block changes to the active state and starts processing.

4.3 i_stParamInputs (STRUCT)

- sServerIpAdr: IP address or hostname of the server to which the client wants to establish a connection.
- udiPortNr: TCP port for communication. Default 502.
- tTimeoutMbRequest: Timeout after a request was sent from the Client/Master but no response was received.

4.4 Inputs

- i_enFcCode: Modbus function to be executed by the slave. Functions 1,2,3,4,5,6,15,16 are supported by the Client/Master.
- i_uiStartAdr: Modbus data register to be used at the Server/Slave device. This number is transferred to the slave unchanged.
- i_uiQuantData: Number of data words to be read or written in word-oriented Modbus functions (e.g. Function Code 3). At bit-oriented Modbus functions the parameter specify the number of bits (inputs or coils). Each coil is read into a separate BYTE (e.g. Function Code 1).

- **i_usiSlaveAdr:** Modbus (RTU) station address (1...247). The Unit-ID is written with this value. Parameter is required for a Modbus RTU via Modbus TCP connection.
- **iq_arbCoilData:** Memory array for bit-oriented Modbus functions (inputs or coils), during read operations the read data is stored in the array. In case of send actions, the data is transferred from the array to the slave device.
- **iq_arwRegisterData:** Memory array for word-oriented Modbus functions, during read operations the read data is stored in the array. In the case of send actions, the data is transferred from the array to the slave device.

4.5 q_xStandby (BOOL)

Indicates that the function block has been enabled and the parameter check was successful. Function Block in standby state.

4.6 q_xActive (BOOL)

Indicates that the function block is in state active. The output becomes true if the process values are valid and the input **i_xExecute** is active.

4.7 q_xBusy (BOOL)

If true, the FB performs some internal tasks (shutdown or initialization) which may take multiple cycles.

4.8 q_xDone (BOOL)

If the process is completed the output is true. By resetting the input **i_xExecute** it is now possible to switch to the standby state and start the next request.

4.9 q_xError (BOOL)

Indicates that an error occurred.

4.10 q_stError (STRUCT)

Displays error information in case of disturbed or faulty communication.

5 Create a Modbus TCP communication

1. Add the Modbus TCP Client library **libWIModbusTCPClient** to your PLC project.
2. It is **essential** to know the parameters and settings of the Modbus Server/Slave. The Modbus communication parameters must be adjusted.
In this Application Note a free Modbus TCP Server simulation Software (ModRSsim2) is used²:

Example:

IP-Address Slave: 192.168.101.40
Port: 502
Function Code: ET_ModbusTCP_FunctionCode.enReadHolReg

3. Import the function Block **FB_ModbusTCPClient** to your PLC program and create a program sequence to execute the function block.

5.1 Example Program

The PLC program shows how it will be possible to create a Modbus TCP communication. In the current example project the function code 3 (Read holding registers) is used.

Explanation of the program flow:

- **step 0:** reset Function Block, and Error Messages and jump to step 10

```
(*Example: Read/Write data to the Modbus slave*)
IF (xModbusCom) THEN

    CASE wModbusCom OF
        0: xEnable := FALSE;
           xExecute:= FALSE;
           strErrorModule := '';
           strErrorSource := '';
           strErrorReason := '';
           strError1 := '';
           strError2 := '';
           strError3 := '';
           wModbusCom:= 10;
```

² Download link: [ModRSsim2](#).

- **step 10:** set the parameters, process values and data and enable the function block. If the function block gives a feedback that the parameters are valid (q_xStandby), start **Execute** and jump to next step.

If an error occurs jump to step 99.

```

10: xEnable          := TRUE;
   enFunctionCode    := ET_ModbusTCP_FunctionCode.enReadHoiReg;      (* FC code 3*)
   uiStartAdr        := 0;                                           (* Start Register *)
   uiQuantData       := 10;                                          (* Lenght *)
   usiSlaveAdr       := 3;                                           (* Slave Address for modbus RTU network*)
   stParameterInputs.sServerIpAdr    := '192.168.101.40';           (* IP Address MB Server*)
   stParameterInputs.udiPortNr       := 502;                        (* Port MB Server*)
   stParameterInputs.tTimeoutMbRequest := T#2S;                    (* Timeout Error *)

IF( enFunctionCode = 5 OR enFunctionCode = 6) THEN
  arbCoilData[1] := TO_BYTE(TRUE);
  arwRegisterData[1] := 125;
ELSIF( enFunctionCode = 15 OR enFunctionCode = 16) THEN
  arbCoilData[1] := TO_BYTE(TRUE);      (*Example Data*)
  arbCoilData[2] := TO_BYTE(TRUE);
  arbCoilData[3] := TO_BYTE(FALSE);
  arbCoilData[4] := TO_BYTE(TRUE);
  arbCoilData[5] := TO_BYTE(TRUE);
  arwRegisterData[1] := 125;
  arwRegisterData[2] := 5000;
  arwRegisterData[3] := 2600;
  arwRegisterData[4] := 8900;
  arwRegisterData[5] := 2300;
END_IF

IF( xStandby ) THEN
  xExecute := TRUE;
  wModbusCom := 20;
ELSIF( xerror) THEN
  wModbusCom := 99;
END_IF

```

- **Step 20:** If the process data validation is ok (q_xActive), jump to next step. If an error occurs jump to step 99.

```

20: IF( xActive ) THEN
  wModbusCom := 30;
ELSIF( xerror) THEN
  wModbusCom := 99;
END_IF

```

- **Step 30:** When data has been received (q_xDone), data from Modbus Server/Slave registers are available. If the communication should be run permanently, the step sequence jumps to step 10, otherwise the sequence is finished and jumps to step 100 and ends.
If an error occurs jump to step 99.

```

30: IF( xdone )THEN
    xExecute      := FALSE;
    arbCoilDataSave := arbCoilData;
    arwRegisterDataSave := arwRegisterData;
    IF( xPermanent )THEN
        wModbusCom := 10;
    ELSE
        wModbusCom := 100;
    END_IF

    ELSIF( xerror )THEN
        wModbusCom := 99;
    END_IF

```

- **Step 99:** show the error messages

```

99: (*Error handling*)
    strErrorModule := sterror.strModule;
    strErrorSource := sterror.strSource;
    strErrorReason := sterror.strReason;
    strError1 := sterror.arstrParameter[1] ;
    strError2 := sterror.arstrParameter[2] ;
    strError3 := sterror.arstrParameter[3] ;

```

- **Step 100:** End

```

100: xModbusComTRUE := FALSE;

    END_CASE
END_IF

```

5.2 Run Example

- Open the simulation program and change the values at Register 0-9 that should be read by the Modbus client.
- Run the PLC program and observe the behavior of the program.

Note: For debugging it is possible to read the bus communication via the simulation tool. Click on the button in the lower right corner (Comms)

