



u-remote library for TIA Portal V16

Abstract:

This library contains function blocks/functions for easy use of Weidmüller u-remote modules in Siemens TIA Portal.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UR20-1COM-232-485-422	1315750000	-
2	UR20-2AI-SG-24-DIAG	1990070000	-
3	UR20-1COM-232-485-422-V2	2826800000	-
4	UR20-FBC-PN-IRT-V2	2566380000	-
5	UR20-FBC-PN-ECO	2659680000	-

Software reference

No.	Software name	Article No.	Software version
1	TIA Portal	-	V16

File reference

No.	Name	Description	Version
1	-	-	-

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>

Revision history

Library Version	Date	Change log	Author
0.1.0	2019-04	Beta	w010843
0.2.0	2021-05	added FB_UR20_2AI_SG_24_DIAG_Param, FB_UR20_2AI_SG_24_DIAG bugfixes of FB_UR20_1COM_232_485_422_Control	w010485
0.3.0	2021-07	added FB_UR20_ModbusRtuMaster	w010485
0.4.0	2021-10	Minor changes in documentation of FB_UR20_ModbusRtuMaster	w010347
0.5.0	2022-01	Added FB_UR20_DIAG, FB_UR20_1COM_232_485_422_V2_Serial FB_UR20_1COM_232_485_422_V2_ModbusMaster	W011037
1.0.0	2023-08	Changed Versioning on demand Bugfixes FB_UR20_1COM_232_485_422_V2_ModbusMaster Added FB_UR20_1COM_232_485_422_V2_ModbusSlave	w01448
1.0.1	2024-01	Fixed bug in FB_UR20_2AI_SG_24_DIAG_CALIB Fixed bug in FB_UR20_DIAG Minor changes in documentation	w010347
1.0.2	2024-07	Fixed bug in FB_UR20_2AI_SG_24_DIAG_CALIB Extended FB_UR20_2AI_SG_24_DIAG_CALIB Added feature "triggered read" to FB_UR20_1COM_232_485_422_V2_ModbusMaster	w01448

Content

1	Warning and Disclaimer	6
2	Standardized behavior of the function blocks.....	7
2.1	Function block variants	7
2.2	Basic states	7
2.3	Inputs and Outputs.....	7
2.4	Simplified behavior model.....	8
2.5	Usage of the function blocks.....	9
3	FB_UR20_1COM_232_485_422_Control.....	11
3.1	Functional description	11
3.2	Inputs.....	11
3.3	Outputs.....	12
3.4	InOut.....	12
3.5	Possible errors	12
4	FB_UR20_ModBusRtuMaster	13
4.1	Functional description	13
4.2	Inputs.....	13
4.3	Outputs.....	14
4.4	InOut.....	14
4.5	Possible errors	15
5	FB_UR20_2AI_SG_24_DIAG_Param.....	16
5.1	Functional description	16
5.2	Inputs.....	16
5.3	Outputs.....	17
5.4	Possible errors	17
6	FB_UR20_2AI_SG_24_DIAG_Calib	18
6.1	Functional description	18
6.2	Inputs.....	19
6.3	Outputs.....	20
6.4	Possible errors	21
7	FB_UR20_DIAG.....	22
7.1	Functional description	22
7.2	Inputs.....	22
7.3	Outputs.....	22
7.4	Possible errors	23

8	FB_UR20_1ComRs_232_485_422_V2_Serial	24
8.1	Functional description	24
8.2	Inputs.....	24
8.3	Outputs.....	25
8.4	InOut.....	25
8.5	Possible errors	26
9	FB_UR20_1ComRs_232_485_422_V2_ModbusMaster	27
9.1	Functional description	27
9.2	Inputs.....	28
9.3	Outputs.....	28
9.4	InOut.....	29
9.5	Possible errors	30
10	FB_UR20_1ComRs_232_485_422_V2_ModbusSlave	31
10.1	Functional description	31
10.2	Inputs.....	31
10.3	Outputs.....	32
10.4	InOut.....	33
10.5	Possible errors	34

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 provided safety equipment / electrical safety design or other redundant safety features that are independent from the automation system.

Disclaimer

This software (programs, function blocks, functions, etc.) does not relieve you of the obligation to handle it safely during use, installation, operation and maintenance. Every user is responsible for the correct operation of his control system.

By using this software 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

This software does not represent customer-specific solutions, it is simply intended to help for typical tasks. The user is responsible for the proper operation of the used products. This software does not relieve you of the obligation of safe use, installation, operation and maintenance. This software is not binding and do not claim to be complete in terms of configuration as well as any contingencies.

By using this software, you acknowledge that Weidmueller cannot be held liable for any damages beyond the described liability regime. We reserve the right to make changes to this software at any time without notice.

We assume no liability for this software or the information contained in this document. Our liability, for whatever legal reason, for damages caused by the use of the software or instructions described in this document 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 Standardized behavior of the function blocks

All Weidmüller function blocks work in a defined manner. This behavior is defined by an internal state machine, which includes a set of standardized inputs and outputs. The following part describes the basic interfaces and behavior of the function blocks, as well as procedures to activate and deactivate the function blocks and how to handle errors.

2.1 Function block variants

There are in total four different variations of the underlying behavior.

- A level-controlled function block changes state by the level of the input signals (enable and execute)
- An edge-controlled function block changes state by evaluating the positive edges of its control inputs (enable, disable, execute, abort)

Both variants can either be implemented as a finite (with additional output "q_xDone") or as a continuous behavior. The finite behavior is used for any kind of action that comes to a defined end, while the continuous behavior is used for any action of an infinite nature.

2.2 Basic states

There are four basic states of a function block:

- **idle**: no action is performed (initial state of each function block).
- **standby**: the function block is initialized and ready to be executed
- **active**: the function block is performing its intended task
- **error**: an error occurred, and the function block had to be stopped.

The states **standby** and **active** are separated into different sub-states, since the function block may take some time to be initialized (entering state **standby**) or to safely shutdown an action (state **active**).

2.3 Inputs and Outputs

The inputs for the level-triggered behavior:

Input	Description
i_xEnable	Set true to enable the FB and start the initialization. The FB is initialized and ready (standby) if q_xStandby is true and q_xBusy is false . If set to false , all actions are stopped immediately, the FB is disabled and possible errors are reset.
i_xExecute	When the FB is initialized and ready (standby), setting i_xExecute to true starts the intended task and the FB becomes active. The output q_xActive indicates that the FB is being executed. If the FB implements a finite behavior, q_xDone indicates that the task is completed. If set to false , the active FB is stopped in a controlled way and the FB switches back to standby state. The output q_xBusy is true during shutdown.

The inputs for edge-triggered behavior:

Input	Description
-------	-------------

i_xEnable	A rising edge enables the FB and starts the initialization. The FB is initialized and ready (standby) if q_xStandby is true and q_xBusy is false .
i_xExecute	When the FB is initialized and ready (standby), a rising edge starts the intended task and the FB becomes active. The output q_xActive indicates that the FB is being executed. If the FB implements a finite behavior, q_xDone indicates that the task is completed.
i_xAbort	When the FB is active (q_xActive is true), a rising edge stops the FB in a controlled way. The output q_xBusy is true during shutdown. The FB switches back to standby .
i_xDisable	A rising edge disables the FB. All actions are stopped immediately and possible errors are reset.

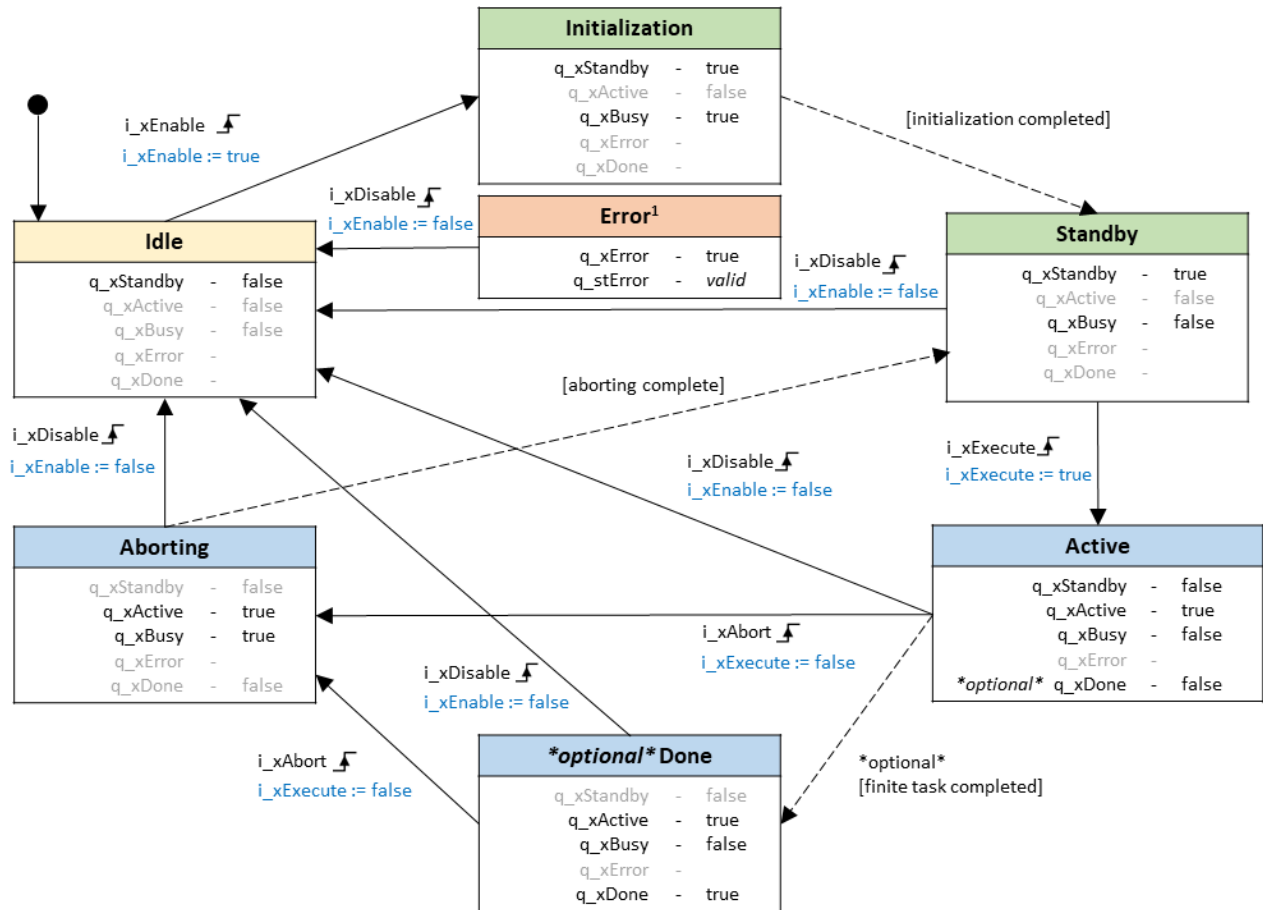
Outputs:

Output	Description
q_xStandby	If true , the FB is either initializing, which may take multiple cycles (output q_xBusy is true), or already initialized and ready to be started (output q_xBusy is false)
q_xActive	If true , the FB is active. The FB is currently performing the desired task (q_xDone and q_xBusy are both false), or has already completed the desired, finite task (q_xDone is true and q_xBusy is false), or is currently shutting down an ongoing task in a controlled manner (q_xBusy is true)
q_xBusy	If true , the FB performs some internal tasks (shutdown or initialization) which may take multiple cycles. Wait until q_xBusy is false . Disabling (via inputs i_xEnable or i_xDisable) is always possible, but bypassing the usual shutdown sequence might result in undesired behaviour (depending on the specific function block).
q_xError	If true , an error has occurred and the FB is stopped.
q_stError	A struct that contains detailed information in case of an error (if q_xError is true). Collecting relevant error details may take some time, so the information may not be available immediately after q_xError indicates an error. Therefore, the output q_stError.xErrorDataValid is set to true as soon as all information is available in q_stError. Prior to that, q_stError might contain outdated or incorrect information.
q_xDone	(Finite behavior only) If true , a finite task has been completed. Reset and back to Standby state (q_xStandby) by disabling i_xExecute.

2.4 Simplified behavior model

Combined view illustrating both level-triggered and edge-triggered behavior is shown on the figure. Blue labels at transitions with inputs correspond to level-triggered behavior¹.

¹ The state "**Error**" is reached from any other state in case of a general error. For reasons of simplicity, the corresponding transitions are not shown in the diagrams.



2.5 Usage of the function blocks

► Activation of the function block

The activation of an FB is controlled by the two boolean inputs of the function block, `i_xEnable` and `i_xExecute`. The activation process is split into two procedures and performed in the same way for every type of behavior:

1. Initialize the function block: switch from state **idle** to **standby**.
By setting `i_xEnable`, parameters are validated, and the block is initialized. The parameter validation takes one single PLC cycle and results either into a start-up routine or an error. If needed, the start-up routine may perform some initial actions, which may take multiple PLC cycles and may also include interaction with devices, parameterization, communication, etc. After the input `i_xEnable` has been set, the output `q_xStandby` changes from **false** to **true**. As long as the function block performs its initialization routine, the output `q_xBusy` stays **true**. After the initialization is finished, `q_xBusy` changes back to **false**, which indicates the function block can be executed now.
2. Execute the function block and perform its intended task: switch from state **standby** to state **active**. Once the output `q_xStandby` indicates **true** and `q_xBusy` has been reset to **false**, the main operation can be started. This is always done by setting the input `i_xExecute` to **true**. The next step (internally performed) is to check whether the current process values are ok. If they are not ok, the activation results in an **error** state. If they are ok, the function block starts with its intended action. While this action is performed, the output `q_xActive` is **true**. After a function block that implements a finite behavior has

performed its action completely, the output `q_xDone` is set to **true**. A continuous type function block stays in state **active** as long as no disable command is given.

► Deactivation of the function block

Once the intended action of the function block has been done (or an ongoing action needs to be aborted), the function block needs to be deactivated. The procedure differs with regard to the implemented behaviour of the function block:

- for a level-controlled function block, the input `i_xExecute` needs to be set to **false**.
- for an edge-controlled function block, a positive edge on the input `i_xAbort` is required.

Once the deactivation command has been received, the function block will change to state **standby**. In some cases, a shutdown routine is implemented and while this routine is executed, the output `q_xBusy` becomes **true** to indicate that the FB is shutting down. After the shutdown procedure has been finished, the outputs `q_xActive` and `q_xBusy` (and possibly `q_xDone`) will change to **false**, and `q_xStandby` will become **true** to indicate that the function block is now again in state **standby**.

A deactivation can also be achieved by

- setting the `i_xEnable` to **false** (in case of a level-controlled function block), or
- providing a rising edge on the input `i_xDisable` (in case of an edge-controlled function block).

This way of deactivating the function block sets it back to its idle state. All outputs are set to **false** and possible errors are reset. This bypasses the usual shutdown sequence and might result in undesired behaviour (depending on the specific function block). For reactivation, the complete activation procedure is required.

► Detect and reset an error

In some cases, errors might occur, and the function block will switch into an **error** state. During this state, the error information is generated and provided via the `q_stError` output. The process of collecting error information is finished once `q_stError.xErrorDataValid` becomes **true**. While this value is **false**, any information contained in `q_stError` is not valid and should not be processed, even if `q_xError` already indicates that an error has occurred. To reset the function block after an error, it needs to be disabled (`i_xEnable` = **false** or positive edge on `i_xDisable`).

3 FB_UR20_1COM_232_485_422_Control

3.1 Functional description

The UR20-1COM-232-485-422 control function block. The function block provides the possibility to send and receive data either on RS232, RS485 or RS422. The parametrization of the module must be done via hardware configuration in the PLC project.

The function block can be used in combination with the following u-remote modules:
UR20-1COM-232-485-422.

3.2 Inputs

Name	Type	Comment
i_hwModule	HW_IO	Hardware ID of UR20-1COM-232-485-422
i_xEnable	BOOL	Enable function block for initialization
i_xExecute	BOOL	Execute action
i_xReadWriteMode	BOOL	Defines the type of operation 0 = write 1 = read
i_iTransactionSize	INT	Number of bytes to be send or read
i_stControlParameter :	STRUCT	Contains parameter for operation
xTxBufferBehavior	BOOL	Defines the behavior of the TX buffer 0 = direct sending 1 = triggered sending
xCreateModbusCRC	BOOL	Activates and deactivates the local CRC check for Modbus RTU
tTransmissionWatchdogTime	TIME	Time to stop ongoing transmission if no response has been received
xTransmissionWatchdogActive	BOOL	Transmission watchdog 0 = inactive 1 = active
tMemoryFlushTimer	TIME	Waiting time for flushing the RX and TX buffer
tMemoryFlushTimeOut	TIME	Timeout flushing memory
xMemoryFlushTimerActive	BOOL	Memory flush timer 0 = inactive 1 = active

3.3 Outputs

Name	Type	Comment
q_xStandby	BOOL	Waiting for activation
q_xActive	BOOL	Function block is activated
q_xBusy	BOOL	Function block is activated and doing its supposed task
q_xDone	BOOL	Execution has come to its supposed end
q_xError	BOOL	Function block is in error state
q_stError :	STRUCT	Detailed error information
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
srtModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [1 .. 3] OF STRING	Additional error information
q_xReceivedNewMessage	BOOL	Message from module new data present
q_xReceivedBufferNearlyFull	BOOL	Message from module only 10 bytes left in buffer

3.4 InOut

Name	Type	Comment
iq_arbReceiveData	ARRAY[*] OF BYTE	Output for all data that shall be read
iq_arbTransmitData	ARRAY[*] OF BYTE	Input for all data that shall be send

3.5 Possible errors

Reason	Description
Communication fault	Communication error detected during operation
Frame to long during read	A frame that was received by the module is too long for being read into the output array of the function block
Frame to long during write	A frame that shall be send is too long for the input buffer of the module
Transmission watchdog exceeded	The module did not response within a specific time during a transmission
Invalid parameter	Watchdog time is too short – select watchdog time > 0
Watchdog for buffer flush expired	Time to flush the buffer was too long

4 FB_UR20_ModBusRtuMaster

4.1 Functional description

The function block provides Modbus RTU master functionality for the UR20-1COM-232-485-422 Serial Module. The FB_UR20_1COM_232_485_422 function block is used internally, so any of its faults may appear. To get more information regarding possible faults of the underlying FB, to refer to section 3.5 of this document. After enabling the function block, it is ready for its action. To send a message, the user needs to set the communication settings like slave address or function code. This is possible after the function block has been enabled or idle. Depending on the function code, the block uses the coil array iq_arxCoilData (FC 1, 2, 5, 15) or register array iq_arwRegisterData (FC 4, 3, 6, 16) to read data to be sent or store data that was received from the slave. Once the execute input i_xExecute is set to **true**, the request to the slave node is sent and the function block waits on its response. Once the response has been received, the function block indicates this by setting the out q_xDone. If there is no response or invalid data from the slave, the function block changes to an **error** state. If the master requested data from the slave, the data is stored into the coil or register array. **Once input i_xExecute is set to false, these arrays are reset to false or zero.** To avoid data loss the array data needs to be saved or processed before. The communication settings can be changed during standby state (i.e. q_xStandby is **true**) before sending the next message.

The function block can be used in combination with the following u-remote modules:
UR20-1COM-232-485-422

4.2 Inputs

Name	Type	Comment
ip_hwHardwareID	HW_IO	Hardware ID of UR20-1COM-232-485-422
i_xEnable	BOOL	Enable function block for initialization
i_xExecute	BOOL	Execute action
i_usiSlaveAdr	USINT	Modbus slave address (1-247)
i_usiFcCode	USINT	Modbus function code (1,2,3,4,5,6,8,15,16)
i_usiStartAdr	USINT	Starting address of register or coils (0-65535)
i_usiQuantityData	USINT	Quantity of registers or coils to be read/written (1-2000 coils or 1-125 registers)
ip_stParameterInputs :	STRUCT	Struct that contains control parameter
tReqTimeout	TIME	Modbus communication timeout after sending a request (default: 2000 ms)

Name	Type	Comment
tHwMemFlushTimer	TIME	Waiting period after flushing the TX/RX buffer inside the UR20-1Com Module (default: 10 ms)
tWaitAfterSend	TIME	Waiting period after sending the request message (default: 20 ms)

4.3 Outputs

Name	Type	Comment
q_xStandby	BOOL	Waiting for activation
q_xActive	BOOL	Function block is activated
q_xBusy	BOOL	Function block is activated and doing its supposed task
q_xDone	BOOL	Execution has come to its supposed end
q_xError	BOOL	Function block is in error state
q_stError :	STRUCT	Detailed error information
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
strModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [1 .. 3] OF STRING	Additional error information

4.4 InOut

Name	Type	Comment
iq_arxCoilData	ARRAY[1..2000] OF BOOL	Read and send buffer for bit-oriented Modbus coils and discrete inputs
iq_arwRegisterData	ARRAY[1..125] OF WORD	Read and send buffer for word-oriented Modbus registers

4.5 Possible errors

Reason	Description
Invalid process value	One of the module parameters is wrong, refer to q_stError.arstrParameter[1], which indicates the faulted parameter input
Invalid parameter	The communication settings are wrong, refer to q_stError.arstrParameter[1], which indicates the faulted parameter input
Timeout 1Com-Driver	Sub-FB doesn't return an answer, please check communication settings with modbus-Slave
Invalid Response	Modbus slave returns an invalid value
Slave Error Response	Modbus slave returns an errorcode

5 FB_UR20_2AI_SG_24_DIAG_Param

5.1 Functional description

The function block can be used to write a parameter set for one of the two input channels of the strain gauge module. This channel is selected by a process input `i_iChannel` and the module itself by its hardware reference `ip_hwHardwareID`, which is an input parameter. The module parameters are implemented as process values. They are checked and saved before writing them to the module and the writing is triggered by the input `i_xExecute`. After a parameter set of a channel is written, the done state of the function block is reached to indicate the successful writing. Any failure results in an **error** state.

The function block can be used in combination with the following u-remote modules:

UR20-2AI-SG-24-DIAG



Be aware that related to the used function WRREC of Simatic it is important to call the instances of this function block sequentially for parametrization of different channels of the UR20-2AI-SG-Module. With using two instances of this block at the same time the risk of misfunction is given in terms of writing wrong parameter.

5.2 Inputs

Name	Type	Comment
<code>ip_hwHardwareID</code>	HW_IO	Hardware ID of UR20-2AI-SG-24-DIAG
<code>i_xEnable</code>	BOOL	Enable function block for initialization
<code>i_xExecute</code>	BOOL	Write parameter set
<code>i_iChannel</code>	INT	Channel of the strain gauge module
<code>i_bConnectionType</code>	BYTE	Connection type 4-Channel (0) / 6-Channel (1)
<code>i_bConversionTime</code>	BYTE	Conversion time according to the u-remote manual: 800 ms (0) / 400 ms (1) / 240 ms (2) / 160 ms (3) / 80 ms (4) / 20 ms (5) / 10 ms (6) / 5 ms (7)
<code>i_bChannelDiagnostic</code>	BYTE	Activate channel diagnostic (0) / deactivate channel diagnostic (1)
<code>i_bTareFunction</code>	BYTE	Set up tare function: deactivate (0) / digital input (1) / software (2) / digital input AND software (3) / digital input OR software (4)
<code>i_diSensorSensitivity</code>	DINT	Sensitivity of the connected sensor: 500,000 ... 960,000,000
<code>i_diFullScale</code>	DINT	Maximum weight that can be loaded to the sensor

Name	Type	Comment
i_diOffset	DINT	Reference weight for calibration (must be smaller than ip_diFullScale)

5.3 Outputs

Name	Type	Comment
q_xStandby	BOOL	Waiting for activation
q_xActive	BOOL	Function block is activated
q_xBusy	BOOL	Function block is activated and doing its supposed task
q_xDone	BOOL	Execution has come to its supposed end
q_xError	BOOL	Function block is in error state
q_stError :	ST_ErrorInfo	Detailed error information
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
srtModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [1 .. 3] OF STRING	Additional error information

5.4 Possible errors

Reason	Description
Invalid process value	One of the module parameters is wrong, refer to q_stError.arstrParameter[1], which indicates the faulted parameter input
Invalid parameter	The hardware reference is on ip_hardwareid is zero
Writing of parameters to module during active failed	Function block WRREC faulted while writing the parameters to the hardware module, for status of the function block refer to q_stError.arstrParameter[1]
Reading of parameters from module during init failed	Function block RDREC faulted while reading the parameters from the hardware module, for status of the function block refer to q_stError.arstrParameter[1]

6 FB_UR20_2AI_SG_24_DIAG_Calib

6.1 Functional description

The function block can be used to calibrate the u-remote strain gauge module. This requires several steps of parametrization and is done individually for each channel, selected by the user via a parameter input. The first step of the sequence is the initial parametrization during standby state, which is indicated by the output `q_xBusy` of the function block while in standby. The operation is triggered via the input `i_xEnable`. During this state, the parameters are read from the module, the initial parameters are integrated into the data and they are written back to the module. Once this is done, the function blocks main operation can be started.

In its first step, the function block adjusts the offset parameter to create the value zero at the channels process value. Therefore, any weight needs to be removed from the sensor. The user task output supports this with the command “remove weight”. Once the weight has been removed, the function block waits for a steady state of the input values within the set Deviation boundaries and after that the calibration can be started via the input `i_xExecute`.

During the calibration process, the user needs to place the reference weight on the sensor, which is supported with the command “place reference weight” by the user task output. After placing the reference weight (defined by its parameter input), the user needs to acknowledge the step with the input `i_xNextStep`. Again, the function block waits for a steady state of the input values. Based on the now measured values, the offset and the sensitivity are corrected. The calibration process is finished once the function block indicates done. As long as this state is not left, the outputs for the corrected sensitivity and offset display the corrected values.



Be aware that related to the used function WRREC of Simatic it is important to call the instances of this function block sequentially for parametrization of different channels of the UR20-2AI-SG-Module. With using two instances of this block at the same time the risk of malfunction is given in terms of writing wrong parameter.

The below table shows the interaction between FB, user and strain gauge:

The function block..	The user..	The strain gauge channel output value
tells the user to “remove weight”		has wrong offset and sensitivity
	removes weight	has wrong offset and sensitivity
takes steady measurement value and calculates interim offset correction		has wrong offset and sensitivity
writes interim offset correction value to the SG module		has value zero
tells user to “place reference weight”		has wrong sensitivity
	places weight and confirms	has wrong sensitivity
takes steady measurement value, calculates new sensitivity and offset and checks ranges		has wrong sensitivity

writes corrected offset and sensitivity values to sg module		is correct
copies corrected values to outputs and indicates that its task is complete		is correct

6.2 Inputs

Name	Type	Comment
ip_hwHardwareID	HW_IO	Hardware ID of UR20-2AI-SG-24-DIAG
i_xEnable	BOOL	Enable function block for initialization
i_xExecute	BOOL	Execute action
i_xNextStep	BOOL	Trigger next step of the calibration process
ip_iChannel	INT	Channel of the strain gauge module
ip_bConnectionType	BYTE	Connection type 4-Channel (0) / 6-Channel (1)
ip_bConversionTime	BYTE	Conversion time according to the u-remote manual: 800 ms (0) / 400 ms (1) / 240 ms (2) / 160 ms (3) / 80 ms (4) / 20 ms (5) / 10 ms (6) / 5 ms (7)
ip_bChannelDiagnostic	BYTE	Activate channel diagnostic (0) / deactivate channel diagnostic (1)
ip_bTareFunction	BYTE	Set up tare function: deactivate (0) / digital input (1) / software (2) / digital input AND software (3) / digital input OR software (4)
ip_bNumberOfSteadyCycles	BYTE	Amount of conversation cycles measurement value to become steady after a weight change
ip_diSteadyDeviationThreshold	DINT	maximum deviation that is accepted as measurement value in steady state
ip_diInitialSensorSensitivity	DINT	Initial sensitivity of the connected sensor: 500,000 ... 960,000,000
ip_diFullScale	DINT	Maximum weight that can be loaded to the sensor
ip_diInitialOffset	DINT	Initial offset value to adjust the zero position
ip_diReferenceWeight	DINT	Reference weight for calibration (must be smaller than ip_diFullScale)

6.3 Outputs

Name	Type	Comment
q_xStandby	BOOL	Waiting for activation
q_xActive	BOOL	Function block is activated
q_xBusy	BOOL	Function block is activated and doing its supposed task
q_xDone	BOOL	Execution has come to its supposed end
q_xError	BOOL	Function block is in error state
q_stError :	STRUCT	Detailed error information
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
srtModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [1 .. 3] OF STRING	Additional error information
q_strUserTask	STRING	Short textual description of required user interaction for next calibration step (remove or place reference weight)
q_diCorrectedSensitivity	DINT	Corrected sensitivity value, valid after successful callibration
q_diCorrectedOffset	DINT	Corrected offset, valid after sucessful callibration

6.4 Possible errors

Reason	Description
Invalid process value	One of the module parameters is wrong, refer to <code>q_stError.arstrParameter[1]</code> , which indicates the faulted parameter input
Invalid parameter	The hardware reference is on <code>ip_hardwareid</code> is zero
Reading of parameters from module during init failed	Function block <code>rdrec</code> faulted while reading the parameters from the hardware module, for status of the function block refer to <code>q_stError.arstrParameter[1]</code>
Writing of parameters to the module during init failed	Function block <code>wrrec</code> faulted while writing the initial parameters to the hardware module during initialization, for status of the function block refer to <code>q_stError.arstrParameter[1]</code>
Writing of parameters for zero adjustment faulted	Function block <code>wrrec</code> faulted while writing the parameters to adjust the channels process value to zero, for status of the function block refer to <code>q_stError.arstrParameter[1]</code>
Calculation of corrected sensitivity or offset failed	The calculation resulted in sensitivity and offset values that are out of range. <code>q_stError.arstrParameter[1]</code> and <code>q_stError.arstrParameter[2]</code> show the calculated values for further trouble shooting.
Writing of corrected parameters failed	Function block <code>wrrec</code> faulted while writing the corrected parameters for offset and sensitivity to the hardware module, for status of the function block refer to <code>q_stError.arstrParameter[1]</code>
Timeout: measurement signal not steady	During function of calibration the input signal did not catch the limiting values (Threshold) within the set time (steady cycles)
Unknown state steady measurement state machine	An error occurred during the steady state sampling. Check <code>q_stError.arstrParameter[1]</code> and <code>q_stError.arstrParameter[2]</code> for further information

7 FB_UR20_DIAG

7.1 Functional description

Each u-remote module can generate diagnostic data, e.g. in the event of a channel error. The diagnostic is always an array of 0..46 bytes. Please take a look at the u-remote manual for further information. The function block can read the diagnostic data from a u-remote module connected to a profinet coupler. The diagnostic functionality must be activated inside the hardware configuration. Without activation the function block returns an error message. The function block reads all available messages at once. The output array q_arstDeviceDiagData shows the diagnostic messages.

7.2 Inputs

Name	Type	Comment
i_hwID	HW_IO	Hardware ID of the module to be read
i_xEnable	BOOL	Enable function block for initialization
i_xExecute	BOOL	Execute action

7.3 Outputs

Name	Type	Comment
q_xStandby	BOOL	Waiting for activation
q_xActive	BOOL	Function block is activated
q_xBusy	BOOL	Function block is activated and doing its supposed task
q_xDone	BOOL	Execution has come to its supposed end
q_xError	BOOL	Function block is in error state
q_stError :	STRUCT	Detailed error information
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
srtModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [1 .. 3] OF STRING	Additional error information
q_arstDeviceDiagData	ARRAY [0 .. 31] OF STRUCT	Short textual description of required user interaction for next calibration step (remove or place reference weight)

7.4 Possible errors

Reason	Description
Invalid hardware ID	Input device could not be found.
Invalid process value	One of the module parameters is wrong, refer to q_stError.arstrParameter[1], which indicates the faulted parameter input
Reading of parameters from module during init failed	Function block rdrec faulted while reading the parameters from the hardware module, for status of the function block refer to q_stError.arstrParameter[1]

8 FB_UR20_1ComRs_232_485_422_V2_Serial

8.1 Functional description

The function block provides an interface to the UR20-1Com-RS-232-485-422-V2 module in the custom mode configuration. If the module is configured as ModBus RTU master/slave or as DMX master/slave, another function block needs to be used. The data exchange between the application and the function block is implemented as two arrays from variable sizes (iq_arbTransmitData, iq_arbReceiveData). The mapping of the process data is done via two fixed size arrays (i_arbProcessInputData, q_arbProcessDataOutputData). During operation the following modes can be used:

1. Writing (i_uiTransactionSize defines the bytes to write)
2. Reading a fixed number of cycles with full data length (i_uiTransactionSize defined the cycles to read, every cycle can read 16 bytes, as long as there is data in the buffer)
3. Reading the whole buffer up to the end
4. Reading of a single telegram as it is defined in the hardware configuration

The mode is selected by an integer input (i_iSendReceiveMode), the possible values are listed above. The telegram configuration is done in the hardware configuration of the module. Further details can be found in the u-remote manual. Once the function block changes from idle to standby, all the internal buffers of the module are erased. Any read or write cycle is than triggered by the execute input and requires the right mode selection. The selected mode is saved and remains constant, until the transaction is finished, which is indicated by the done signal. As the data input is an array from variable size, it cannot be saved at the beginning of the execution, so the user needs to keep the data constant until the done signal is set.

8.2 Inputs

Name	Type	Comment
ip_hwHardwareID	HW_IO	Hardware ID of the module to be read
i_xEnable	BOOL	Enable function block for initialization
i_xExecute	BOOL	Execute action
i_iSendReceiveMode	INT	Defines the type of operation 0= Transmit 1= recive amount of cycles 2= receive telegram# 3= read buffer
i_iTransactionSize	INT	Number of bytes to be send or read / cycles to read
ip_stWatchDogParameter :	STRUCT	Contains time parameter for operation
tWatchDogDataExchange	TIME	Time to perform the data exchange
tWatchDogShutdown	TIME	Time to shutdown the module

Name	Type	Comment
tWatschDogInitailize	TIME	Time to initialize the module

8.3 Outputs

Name	Type	Comment
q_xStandby	BOOL	Waiting for activation
q_xActive	BOOL	Function block is activated
q_xBusy	BOOL	Function block is activated and doing its supposed task
q_xDone	BOOL	Execution has come to its supposed end
q_xError	BOOL	Function block is in error state
q_stError :	STRUCT	Detailed error information
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
srtModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [0 .. 2] OF STRING	Additional error information
q_xRxBufferNotEmpty	BOOL	The receive buffer of the module contains data that has not been read
q_xRxBufferFull	BOOL	The receive buffer of the module can not take any more data, data loss possible
q_xTxBufferNotEmpty	BOOL	The transmit buffer of the module contains data that has not been transmitted
q_xTxBufferFull	BOOL	The transmit buffer of the module can not take any more data, data loss possible
q_uiBytesRead	UINT	The amount of bytes read during this transaction

8.4 InOut

Name	Type	Comment
iq_arbTransmitData	ARRAY[*] OF BYTE	The data to be transmitted to the module
iq_arbReceiveData	ARRAY[*] OF BYTE	The received data from the module

8.5 Possible errors

Reason	Description
Input data conversion	The input data is corrupted and cannot be processed further. It should be considered to check the mapped data from module
Output data conversion	The output data is corrupted. The error is internal can not be fixed by the user. Weidmüller service needs to be contacted.
Writing procedure to long, watchdog exceeded	No response from the module within timeframe during the writing sequence.
Reading procedure to long, watchdog exceeded	No response from the module within timeframe during the reading sequence.
Shutdown procedure to long, watchdog exceeded	No response from the module within timeframe during the shutdown sequence.
No response while clearing the module	No response from the module while the receive and transmit buffers are flushed
frame to long during read	The frame that is read from the module is longer than the input array. The problem can be solved by extending iq_arbReceiveData, if possible.

9 FB_UR20_1ComRs_232_485_422_V2_ModbusMaster

9.1 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 254 modbus master channels for data exchange. 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 with a variable array named `iq_arstModbusChannels`. To use it, the user needs to define its size. Each communication channel needs an array element. If, for example, the user wants to use three channels of the module, `iq_arstModbusChannels` must be dimensioned as e.g. `[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 to the UR20_1COM_232_485_422_V2.

A read data channel cyclically reads the data specified by `uiOffsetRead` and `uiLengthRead` in `iq_arstModbusChannels[<channel number>]` and stores them in `arwRegisterData` or `arxCoilData` in `iq_arstModbusChannels[<channel number>]`. 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.

On demand: set `iq_arstModbusChannels[<channel number>].uiCycleTime` to zero. The UR20_1COM_232_485_422_V2 module will only send data when the FB_UR20_1ComRs_232_485_422_V2 transfers new data to the module, i.e., when the associated `xTriggerTransaction` flag is set.

After the communication channels were successfully parametrized, the function block confirms by setting `q_xStandby` and waits for the execute signal `i_xExecute` to start its main operation. While the block is active, the channel configuration cannot be changed. The FB starts the Modbus communication mode of the UR20_1COM_232_485_422_V2 module, then it cyclically iterates over the configured Modbus channels:

For those channels with a read configuration, the FB transfers channel read data from the UR20_1COM_232_485_422_V2 module into `arwRegisterData` or `arxCoilData` in `iq_arstModbusChannels`.

For those channels with a write configuration, the FB checks if `iq_arstModbusChannels[<channel number>].xTriggerTransaction` is set. If so, it transfers `arwRegisterData` or `arxCoilData` in `iq_arstModbusChannels[<channel number>]` to the associated channel in the UR20_1COM_232_485_422_V2 module and clears the associated `xTriggerTransaction` flag.

The behavior is depending on the Modbus function codes that the user has provided in the channel configuration. Some function codes exhibit write behavior and others exhibit read behavior. Thus, some channels read data from, and others write data to the Modbus slave device(s) connected to the UR20_1COM_232_485_422_V2 module. If there are both write- and read channels, the FB will alternate between processing a write channel, then a read channel, then a write channel and so on.

To send new cyclic data or any on-demand data, write a new payload to `arwRegisterData` or `arxCoilData` in `iq_arstModbusChannels`, or just keep the old payload. Set `iq_arstModbusChannels[<channel number>].xTriggerTransaction` to true. The function block finishes any ongoing update of a read channel's data, then it writes the new data to the module. After the FB has written the new data to the module, it sets `iq_arstModbusChannels[<channel number>].xTriggerTransaction` to false. For channels that are configured as "write on demand", the UR20_1COM_232_485_422_V2 module will send their data after `iq_arstModbusChannels[<channel number>].xTriggerTransaction` is set to true and the FB has written the data to the module. If the transmit buffer is empty, the UR20_1COM_232_485_422_V2 module will send the data right away. If another channel is transmitting cyclic data, the UR20_1COM_232_485_422_V2 module will send the on-demand data as soon as the ongoing transmit has finished. With activation of triggered reading function (`iq_arstModbusChannels[<channel number>].uiCycleTime` set to "0") the specific channel is only read once the `iq_arstModbusChannels[<channel number>].xTriggerTransaction` is set to true.

The timing of the data transfer depends on the data size and on the cycle time of the field bus.

9.2 Inputs

Name	Type	Comment
<code>ip_hwHardwareID</code>	HW_IO	Hardware ID of the module to be read
<code>i_xEnable</code>	BOOL	Enable function block for initialization
<code>i_xExecute</code>	BOOL	Execute action

9.3 Outputs

Name	Type	Comment
<code>q_xStandby</code>	BOOL	Waiting for activation
<code>q_xActive</code>	BOOL	Function block is activated
<code>q_xBusy</code>	BOOL	Function block is activated and doing its supposed task
<code>q_xDone</code>	BOOL	Execution has come to its supposed end
<code>q_xError</code>	BOOL	Function block is in error state
<code>q_stError :</code>	STRUCT (ST_UR20_1ComRs_232_485_422_V2_ModbusMasterError)	Detailed error information

Name	Type	Comment
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
strModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [0 .. 2] OF STRING	Additional error information

9.4 InOut

Name	Type	Comment
iq_arstModbusChannels:	ARRAY[*] OF STRUCT <small>(ST_UR20_1ComRs_232_485_422_V2_ModbusMasterChannel)</small>	Contains the channel configuration, control bits and transaction data
uiCycleTime	UINT	Configured cycle time in ms to trigger channel operation internally in the module; Cycle time value of "0" activates triggered reading function
uiSlaveAddress	UINT	Address (1-247) for slave device the channel communicates with
iFunctionCode	INT	Configured function code for this channel
uiOffsetWrite	UINT	Register or coil offset that is used by this channel
uiLengthWrite	UINT	Register (1-125) or coil (1-2000) written by the channel
uiOffsetRead	UINT	Register or coil offset that is used by this channel
xTriggerTransaction	BOOL	Triggers the reading or writing transaction depending on the module configuration
iChannelDiag	INT	Diagnostic data of the channel
arwRegisterData	ARRAY [0 .. 124] OF WORD	Stores the transaction data if one of the analog function codes is configured for this channel
arxCoilData	ARRAY [0 .. 1999] OF WORD	Stored the transaction data if one of the binary function codes is configured for this channel

9.5 Possible errors

Reason	Description
Input data conversion	The input data is corrupted and cannot be processed further. It should be considered to check the mapped data from module
Output data conversion	The output data is corrupted. The error is internal can not be fixed by the user. Weidmüller service needs to be contacted.
No response while clearing or configuring the module	Before the module can be used, all channels are configured by this fb. the module stopped responding during this procedure.
Reading/writing procedure to long, watchdog exceeded	The module stopped the communication during reading and writing the channels. The modbus may continue running, as the error only addresses the plc to module communication.
Shutdown procedure to long, watchdog exceeded	The module stopped the communication during the shutdown sequence. The modbus may continue running, as the error only addresses the plc to module communication.
Wrong channel response during read	At the beginning of every readout of a channel, the module writes the current channelnumber into the datastream. during this procedure an error happened.
Read function code occurred during write sequence	Some how a read functioncode was found durement a write sequence. Internal error.
Difference in channel configuration between module and fb detected	The check sequence after channel configuration returned a wrong amount of configured channels.

10 FB_UR20_1ComRs_232_485_422_V2_ModbusSlave

10.1 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 the 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 entry into its state standby by setting output `q_xStandby` to true. Next, it waits for the 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[<channel number>].wRegisters` to the module.

In ongoing active state, the function block cyclically 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 set to 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 it 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 complete, the function block sets the `xTriggerTransaction` flag for that channel back to false.

To use Modbus function codes FC1, FC3, FC5, FC6, FC15, FC16, FC23 activate the following flags in the channel configuration: `iq_arstModbusChannels[<channel number>].xRead` and

`iq_arstModbusChannels[<channel number>].xWrite`.

To use Modbus function code FC2, FC4, enable `iq_arstModbusChannels[<channel number>].xRead only`.

10.2 Inputs

Name	Type	Comment
<code>ip_hwHardwareID</code>	HW_IO	Hardware ID of the module to be read

Name	Type	Comment
i_xEnable	BOOL	Enable function block for initialization
i_xExecute	BOOL	Execute action
ip_stWatchDogParameter :	STRUCT	Contains time parameter for operation
tWatchDogDataExchange	TIME	Time to perform the data exchange
tWatchDogShutdown	TIME	Time to shutdown the module
tWatschDogInitailize	TIME	Time to initialize the module

10.3 Outputs

Name	Type	Comment
q_xStandby	BOOL	Waiting for activation
q_xActive	BOOL	Function block is activated
q_xBusy	BOOL	Function block is activated and doing its supposed task
q_xDone	BOOL	Execution has come to its supposed end
q_xError	BOOL	Function block is in error state
q_stError :	STRUCT	Detailed error information
xErrorDataValid	BOOL	Detailed error information is complete and ready to be read
strModule	STRING	Name of the module / fb
strSource	STRING	Source of the module / fb / sub-fb
strReason	STRING	Description of the error
arstrParameter	ARRAY [0 .. 2] OF STRING	Additional error information
dtTimestamp	DTL	Timestamp of Error occurence

10.5 InOut

Name	Type	Comment
iq_arstModbusChannels:	ARRAY[*] OF STRUCT	Contains the channel configuration, control bits and transaction data
uiSlaveAddress	UInt	Address (1-247) for slave device the channel communicates with
uiOffset	UInt	register or coil offset that is used by this channel
uiLength	UInt	register (1-125) or coil (1-2000) written/read by the channel
xWrite	Bool	register or coil specified gets write permission (refer to Modbus Spec.)
xRead	Bool	register or coil specified gets read permission (refer to Modbus Spec.)
xTransactionRead	Bool	register (1-123) or coil (1-1968) read from 1COM Module (if set to True) or written to 1COM Module (if set to false)
xTriggerTransaction	Bool	triggers the reading or writing transaction depending on the module configuration
arwRegisterData	ARRAY [0 .. 124] OF WORD	stores the transaction data if one of the analog function codes is configured for this channel

10.6 Possible errors

Reason	Description
Input data conversion	The input data is corrupted and cannot be processed further. It should be considered to check the mapped data from module
Output data conversion	The output data is corrupted. The error is internal can not be fixed by the user. Weidmüller service needs to be contacted.
No response while clearing or configuring the module	Before the module can be used, all channels are configured by this fb. the module stopped responding during this procedure.
Reading/writing procedure to long, watchdog exceeded	The module stopped the communication during reading and writing the channels. The modbus may continue running, as the error only addresses the plc to module communication.
Shutdown procedure to long, watchdog exceeded	The module stopped the communication during the shutdown sequence. The modbus may continue running, as the error only addresses the plc to module communication.
Wrong channel response during read	At the beginning of every readout of a channel, the module writes the current channelnumber into the datastream. during this procedure an error happened.
Difference in channel configuration between module and fb detected	The check sequence after channel configuration returned a wrong amount of configured channels.