

u-remote library for CODESYS 3.5

CODESYS package

Abstract:

The library package contains function blocks and functions for easy use of Weidmüller u-remote modules. This document explains the usage of function blocks, functions, and data structures.

Hardware reference

No.	Component name	Article No.	Hardware / Firmware version
1	UR20-1COM-232-485-422-V2	2826800000	-
2	UR20-1COM-232-485-422 V1	1315750000	FW 1.0.0.16
3	UR20-4DI-P	1315170000	-
4	UR20-4DI-P-3W	2009360000	-
5	UR20-4DI-P-3W	2009360000	-
6	UR20-8DI-P-2W	1315180000	-
7	UR20-8DI-P-3W	1394400000	-
8	UR20-8DI-P-3W-HD	1315190000	-
9	UR20-3EM-230V-AC	2007420000	-
10	UR20-4COM-IO-LINK	1315740000	-
11	UR20-2AI-SG-24-DIAG	1990070000	FW 1.00.06
12	UR20-1COM-CANOPEN	2489840000	FW 1.0.0
13	UR20-2AI-UI-16	2705620000	-
14	UR20-4AI-UI-12	1394390000	-
15	UR20-4AI-UI-16	1315620000	-
16	UR20-4AI-UI-16-HD	1506920000	-
17	UR20-2AI-UI-16-DIAG	2566090000	-
18	UR20-4AI-UI-16-DIAG	1315690000	-
19	UR20-4AI-UI-16-DIAG-HD	1506910000	-
20	UR20-4AI-RTD-DIAG	1315700000	-
21	UR20-4AI-RTD-HP-DIAG	2456540000	-
22	UR20-1COM-CAN	3040800000	FW 1.0.0

Software reference

No.	Software name	Article No.	Software version
1	CODESYS	-	V3.5 SP21 or higher
2	CODESYS Control SL for Weidmueller u-OS	-	V4.12.1.0 or higher

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
1.0.0	2021-06	Draft	w010521
1.0.0	2022-01	added libWiUr203EM_230V_AC	w010174
2.0.0	2022-03	Updated libWiUr203EM_230V_AC	w010174
2.0.1	2022-04	Added libWiUr204COM_IOLINK	w010174
2.0.1	2022-06	Added libWiUr20Diag	w010521
2.1.0	2022-10	Updated libWiUr20ComRs_232_485_422_V2	w010174
2.1.x	2022-11	Added libWiUr20AiSg	w010174
2.2.x	2023-07	Updated 1COMv1 and ModBusRTU Master	w010174
2.2.x	2024-02	Added libWiUr20ComCanOpen and Structs	w010521
2.2.x	2024-03	Fixed 2AI-SG sensitivity upper boundary	W010174
2.2.x	2024-07	Added libWiUr20AiUi, libWiUr20RTD, libWiUr20TC, libWiUr20PWM, updated libWiUr20ComRs_232_485_422, libWiUr20ComRs_232_485_422_V2, libWiUr20Diag, libWiUr20_2AI_SG, libWiBehaviorModels, update libWiUr20ComRS_V2 fixed several EcCAN param r/w FB descriptions, abstracted EcCAN- and SysBus param r/w FB descriptions into global and specific chapters	W010174, WM02363
2.2.x	2025-01	Update of libWiUr20ComCanOpen	W010521
3.0.0	2025-04	Breaking change: resolution of library due to change in company name (see chp. 2) Added libWiUr20ComCAN updated libWiUr20ComCANopen Fixed write channel diag missing bug and added xNewDataArrived flag in Modbus Master FB in libWiUr20ComRs_232_485_422_V2	W010174

Content

1	Warning and Disclaimer.....	8
2	libWi library usage	9
2.1	Recommendation for updates from library package versions 2.x.	9
2.2	Issues with unresolved libraries in CODESYS	9
2.3	How to fix unresolved libraries in CODESYS	11
2.4	How to fix unresolved libraries in CODESYS, alternative approach	13
3	Standardized behavior of the function blocks.....	15
3.1	Function block variants	15
3.2	Basic states	15
3.3	Inputs and Outputs	15
3.4	Simplified behavior model.....	17
3.5	Usage of the function blocks.....	17
3.6	Error information handling.....	19
4	Module parametrization	20
4.1	System Bus	20
4.2	EtherCAT.....	23
5	Diagnosis alarm- and process alarm handling	25
5.1	Diagnostic alarm messages.....	25
5.2	Process alarm messages.....	26
6	libWiUr20ComRs_232_485_422.....	28
6.1	FB_UR20_1ComRs_232_485_422_Control	28
6.2	FB_UR20_1ComRs_232_485_422_ParamEcCanRead	31
6.3	FB_UR20_1ComRs_232_485_422_ParamEcCanWrite	33
6.4	FB_UR20_1ComRs_232_485_422_ParamSysBusRead.....	34
3.1	FB_UR20_1ComRs_232_485_422_ParamSysBusWrite.....	34
7	libWiUr20ModbusRTUMaster	36
7.1	FB_ModbusRTUMaster	36
8	libWiUr20Digital.....	40
8.1	FB_UR20_4DI_ParamEcCanRead.....	40
8.2	FB_UR20_4DI_ParamEcCanWrite	41
8.3	FB_UR20_8DI_ParamEcCanRead.....	42

8.4	FB_UR20_8DI_ParamEcCanWrite	43
9	libWiUr20ComRs_232_485_422_V2	45
9.1	FB_UR20_1ComRs_232_485_422_V2_Serial	45
9.2	FB_UR20_1ComRs_232_485_422_V2_ModbusMaster	48
9.3	FB_UR20_1ComRs_232_485_422_V2_ModbusSlave	52
9.4	FB_UR20_1ComRs_232_485_422_V2_ParamSysBusRead	55
9.5	FB_UR20_1ComRs_232_485_422_V2_ParamSysBusWrite	57
9.6	FC_1COM_V2_RawDiagArray_TO_DiagData	58
9.7	FB_UR20_1ComRs_232_485_422_V2_ModbusRTU_CustomMode	60
10	libWiUr203EM_230V_AC	65
10.1	FB_UR20_3EM_Control	65
10.2	UR20_3EM_ChannelParamEcCanWrite	70
10.3	FB_UR20_3EM_ParamEcCanRead	72
10.4	FB_UR20_3EM_ParamEcCanWrite	73
11	libWiUr20Diag	76
11.1	FB_UR20_Uremote_Diagdata_Ethercat	76
12	libWiUr204COM_IOLINK	78
12.1	FB_UR20_4COM_IO_LINKDevice_ParamEcCanRead	78
12.2	FB_UR20_4COM_IO_LINKDevice_ParamEcCanWrite	80
12.3	FB_UR20_4COM_IO_LINKDevice_ParamModbusRead	82
12.4	FB_UR20_4COM_IO_LINKDevice_ParamModbusWrite	84
13	libWiUr20AiSg	87
13.1	FB_UR20_2AI_SG_24_DIAG_ParamEcCanRead	87
13.2	FB_UR20_2AI_SG_24_DIAG_ParamEcCanWrite	88
13.3	FB_UR20_2AI_SG_24_DIAG_ParamSysBusRead	90
13.4	FB_UR20_2AI_SG_24_DIAG_ParamSysBusWrite	91
13.5	FB_UR20_2AI_SG_24_DIAG_Calib	93
13.6	FC_2AI_SG_24_DIAG_RawDiagArray_TO_DiagData	98
14	libWiUr20ComCanOpen	100
14.1	FB_UR20_1Com_CANOpen_Transparent	100
14.2	FB_UR20_1Com_CANOpen_SDORRead_EC	104
14.3	FB_UR20_1Com_CANOpen_SDOWrite_EC	105
14.4	FB_UR20_1Com_CANOpen_SDORRead_PN	107

14.5	FB_UR20_1Com_CANOpen_SDOWrite_PN	109
15	libWiUr20ComCan	111
15.1	FB_UR20_1Com_CAN_Transparent	111
16	libWiUr20AiUi	115
16.1	FB_UR20_AI_UI_ParamSysBusRead	115
16.2	FB_UR20_AI_UI_ParamSysBusWrite	116
16.3	FB_UR20_AI_UI_DIAG_ParamSysBusRead	117
16.4	FB_UR20_AI_UI_DIAG_ParamSysBusWrite	119
16.5	FC_xAI_UI_RawDiagArray_TO_DiagData	119
17	libWiUr20RTD	121
17.1	FB_UR20_4AI_RTD_DIAG_ParamSysBusRead	121
17.2	FB_UR20_4AI_RTD_DIAG_ParamSysBusWrite	122
17.3	FB_UR20_4AI_RTD_HP_DIAG_ParamSysBusRead	122
17.4	FB_UR20_4AI_RTD_HP_DIAG_ParamSysBusWrite	124
17.5	FC_4AI_RTD_Raw_TO_ProcessAlarmData	124
17.6	FC_4AI_RTD_RawDiagArray_TO_DiagData	125
18	libWiUr20TC	127
18.1	FB_UR20_4AI_TC_DIAG_ParamSysBusRead	127
18.2	FB_UR20_4AI_TC_DIAG_ParamSysBusWrite	128
18.3	FC_4AI_TC_RawDiagArray_TO_DiagData	128
18.4	FC_4AI_TC_Raw_TO_ProcessAlarmData	129
19	libWiUr20PWM	131
19.1	FB_UR20_2PWM_PN_xA_Control	131
19.2	FB_UR20_2PWM_PN_xA_ParamSysBusRead	133
19.3	FB_UR20_2PWM_PN_xA_ParamSysBusWrite	134
19.4	FB_UR20_2PWM_I_2_5A_2DI_ParamSysBusRead	135
19.5	FB_UR20_2PWM_I_2_5A_2DI_ParamSysBusWrite	136

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 libWi library usage

Installation of the u-remote library for CODESYS 3.5 package is straight-forward, just download and unpack the library archive from the support center on the Weidmueller homepage and double-click the `libWiPackageCodesys_Vx_x_x.package` file.

Please be aware that sometimes updates of namespace, company name or breaking changes in the code occur during ongoing development and maintenance of the u-remote libraries for CODESYS. Depending on the history of the user's system, the CODESYS IDE may be unable to resolve some library references. This chapter is intended to help users prevent or fix such issues.

2.1 Recommendation for updates from library package versions 2.x.

The version 3.0.0 of the Weidmueller *libWiPackageCodesys* package contains breaking changes. The best way to avoid issues resulting from these changes is to uninstall both older versions of the *libWiPackageCodesys* package and the *CODESYS Control SL for Weidmueller u-OS* package before you install version 3.0.0 of the *libWiPackageCodesys* package and the *CODESYS Control SL for Weidmueller u-OS* package version 4.12.1.0.

2.2 Issues with unresolved libraries in CODESYS

The CODESYS library manager resolves library references by a combination of three parameters stored in each library:

- the company name,
- the namespace and
- the version number.

In version 3.0.0 of the *libWiPackageCodesys* package, Weidmueller fixes some issues with company name spelling and -consistency in all libraries:

- In u-remote library packages up to and including version 2.2.3, the company name in the library descriptions was **"Weidmueller"**.
- In u-remote library packages from version 3.0.0 onwards, the company name in the library descriptions is **"Weidmueller Interface GmbH & Co. KG"**

The *CODESYS Control SL for Weidmueller u-OS* package contains a library called *IoDrvUr20Control*. This library is the driver for the System Bus of several u-OS devices in Weidmueller's portfolio. In release version 4.12.1.0 of the *CODESYS Control SL for Weidmueller u-OS* package, Weidmueller fixes the spelling of the company name in the *IoDrvUr20Control* library:

- In versions up to and including 4.12.0.0, the company name of the library was **"Weidmüller Interface GmbH & Co. KG"**
- In versions from 4.12.1.0 onwards, the company name of the library is **"Weidmueller Interface GmbH & Co. KG"**

Because the CODESYS library manager includes the company name in the library resolution process, it considers two libraries with differing company names as unrelated and does not replace a library with company name “A” with a library with company name “B”, even if they have the same namespace and A’s version number is newer than B’s.

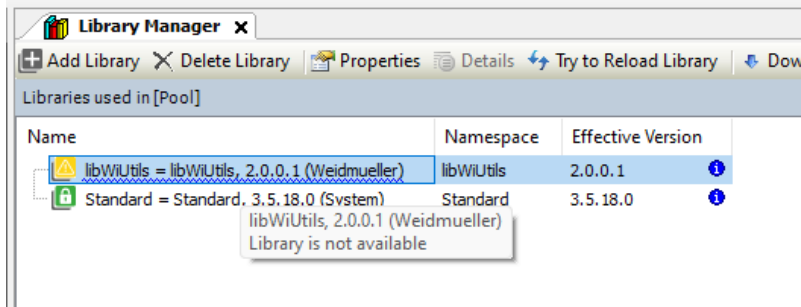
Therefore, the user must update the libraries manually after a company name change.

For the combination of *CODESYS Control SL for Weidmueller u-OS v4.12.1.0* package and the *libWiPackageCodesys v3.0.0*, Weidmueller has already completed this task for you regarding the name change of the *IoDrvUr20Control* library. However, you need to perform the manual library resolution for the u-remote libraries you use *within your projects*.

2.3 How to fix unresolved libraries in CODESYS

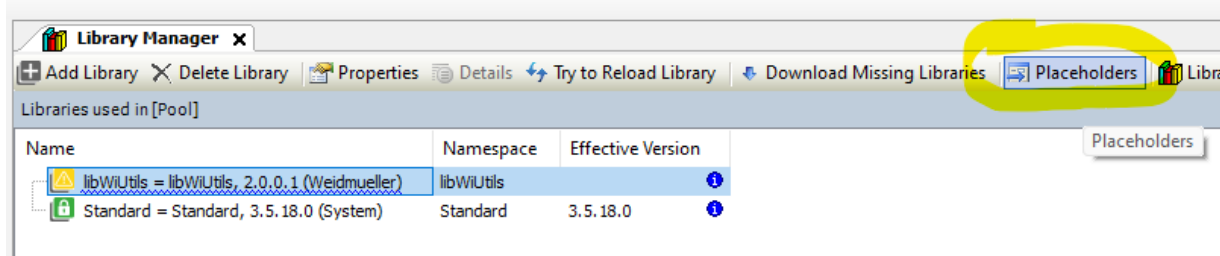
Note that some libraries reference other libraries, themselves. If the Library Manager still indicates a problem with a library after you followed the below steps, check if the library reference you fixed has an issue with a reference to a dependency library and repeat the process, as needed.

Starting with an “unresolved library” compile time error like this:

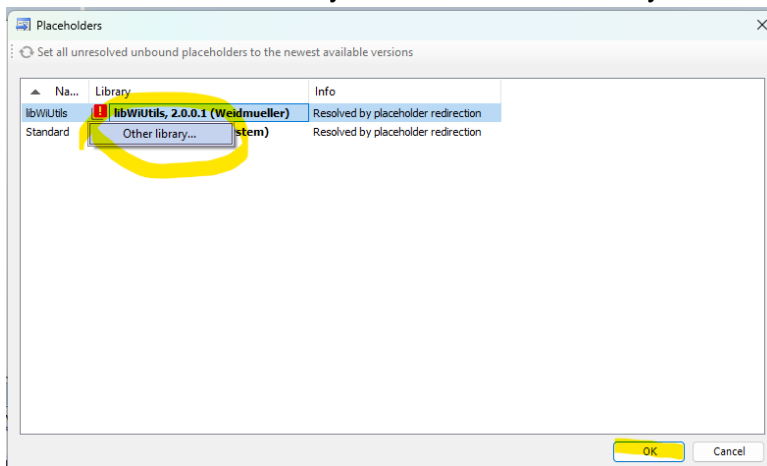


Do the following:

- Select the unresolved library in the Library Manager. Click on “Placeholders” in the menu bar of the Library Manager.

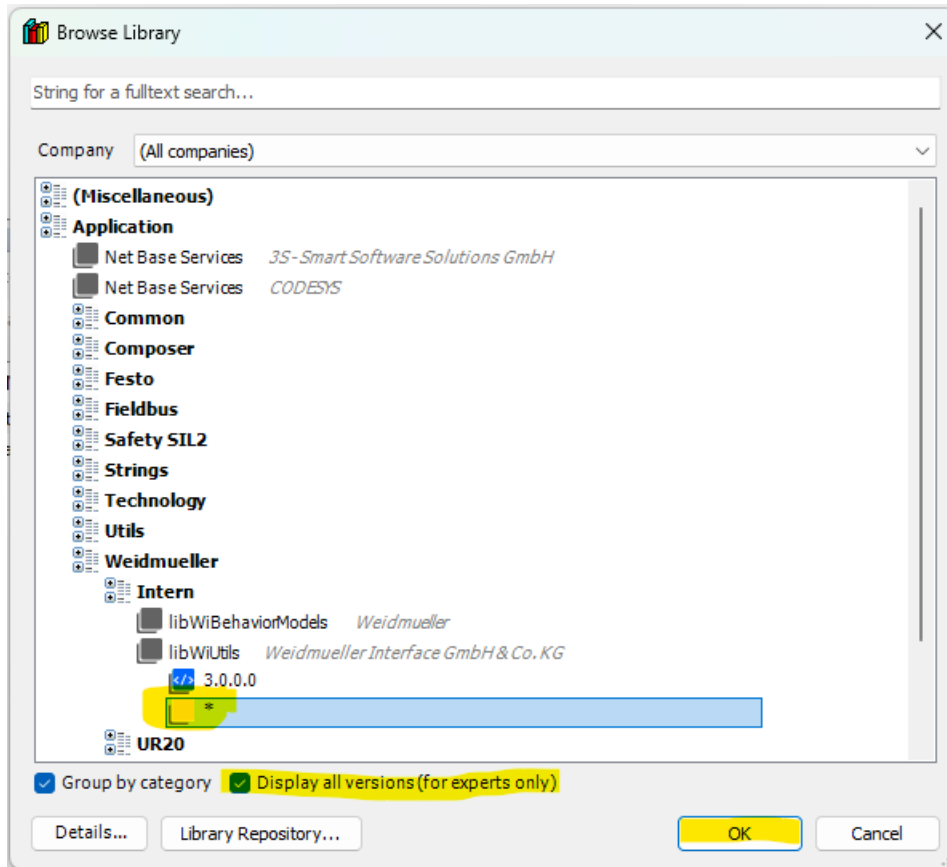


- Click on the library, then click “Other library...” and click “OK”.



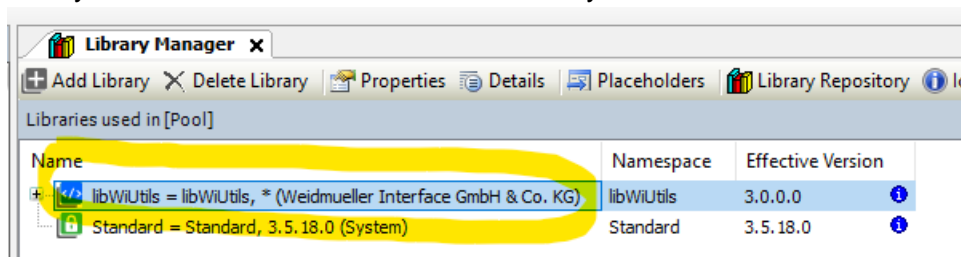
u-remote library for CODESYS 3.5

- Select the “Display all versions (for experts only)” checkbox, then select the needed library version or select the “*” for the asterisk reference. The asterisk reference instructs the Library Manager to reference the library with the newest version number, and matching company name and namespace from your library repository. Click Ok.



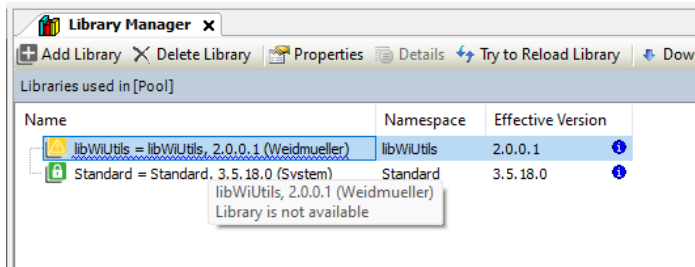
- The CODESYS IDE presents the “Placeholders” dialogue again. Click on “Ok”.

Now you have resolved the unresolved library reference:



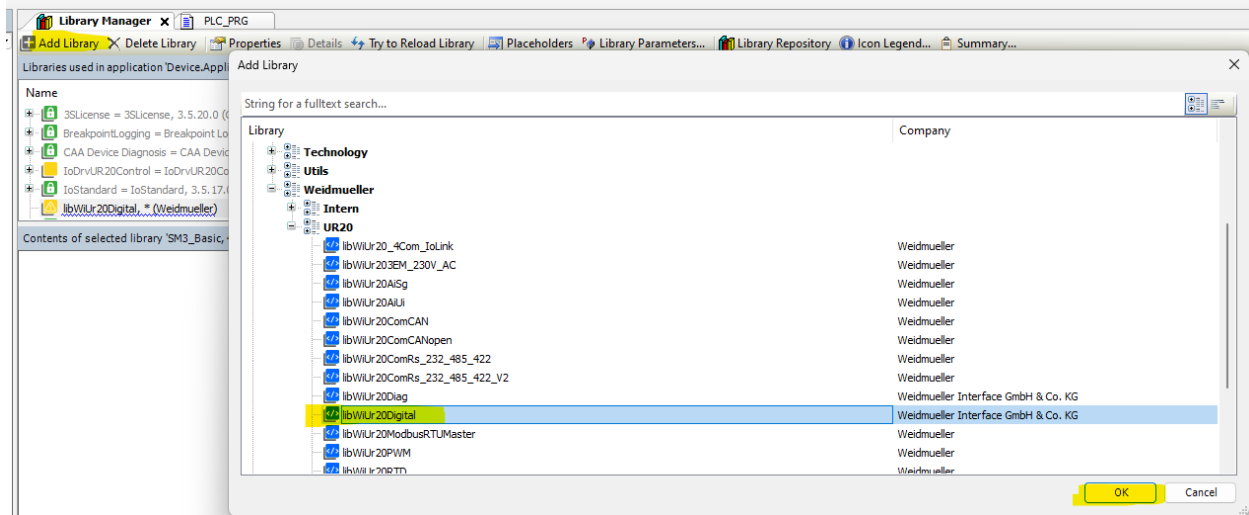
2.4 How to fix unresolved libraries in CODESYS, alternative approach

Starting with an “unresolved library” compile time error like this:

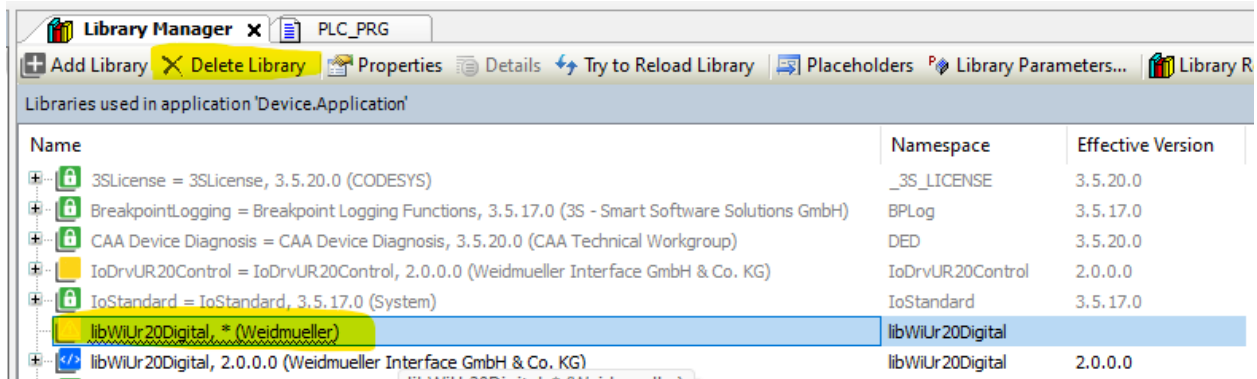


Do the following:

- Click on “Add Library”
- Navigate to the library with the correct name
- Click “OK”



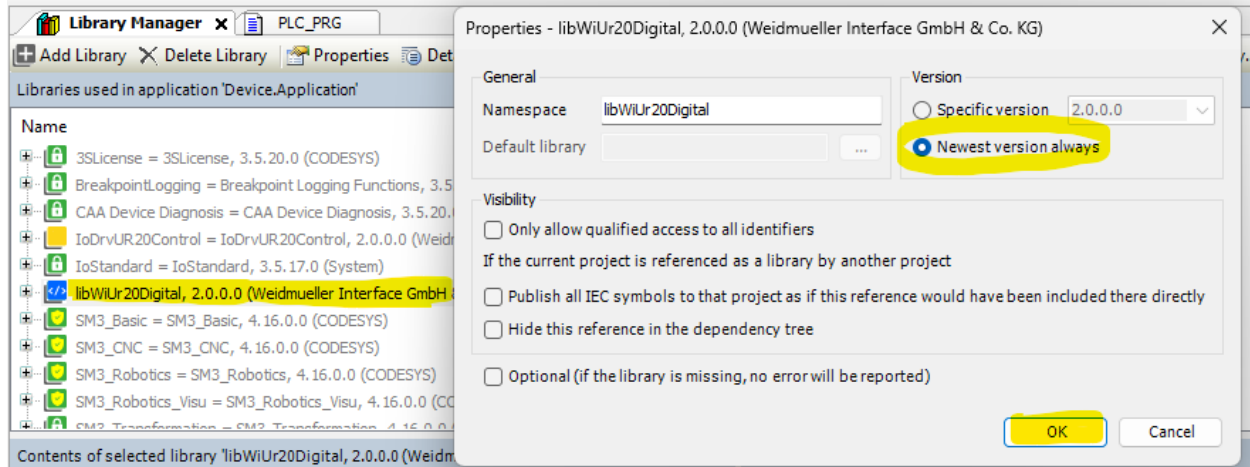
- Select the unresolved library and click “Delete Library”.



u-remote library for CODESYS 3.5

Optionally, configure the correct library for asterisk (“ * ”) reference, to let the Library Manager always use the newest library:

- Right-click on the library, then select “Properties...”
- In the Properties dialogue on the upper right, select “Newest version always” and click “OK”.



3 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.

3.1 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.

3.2 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**).

3.3 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:

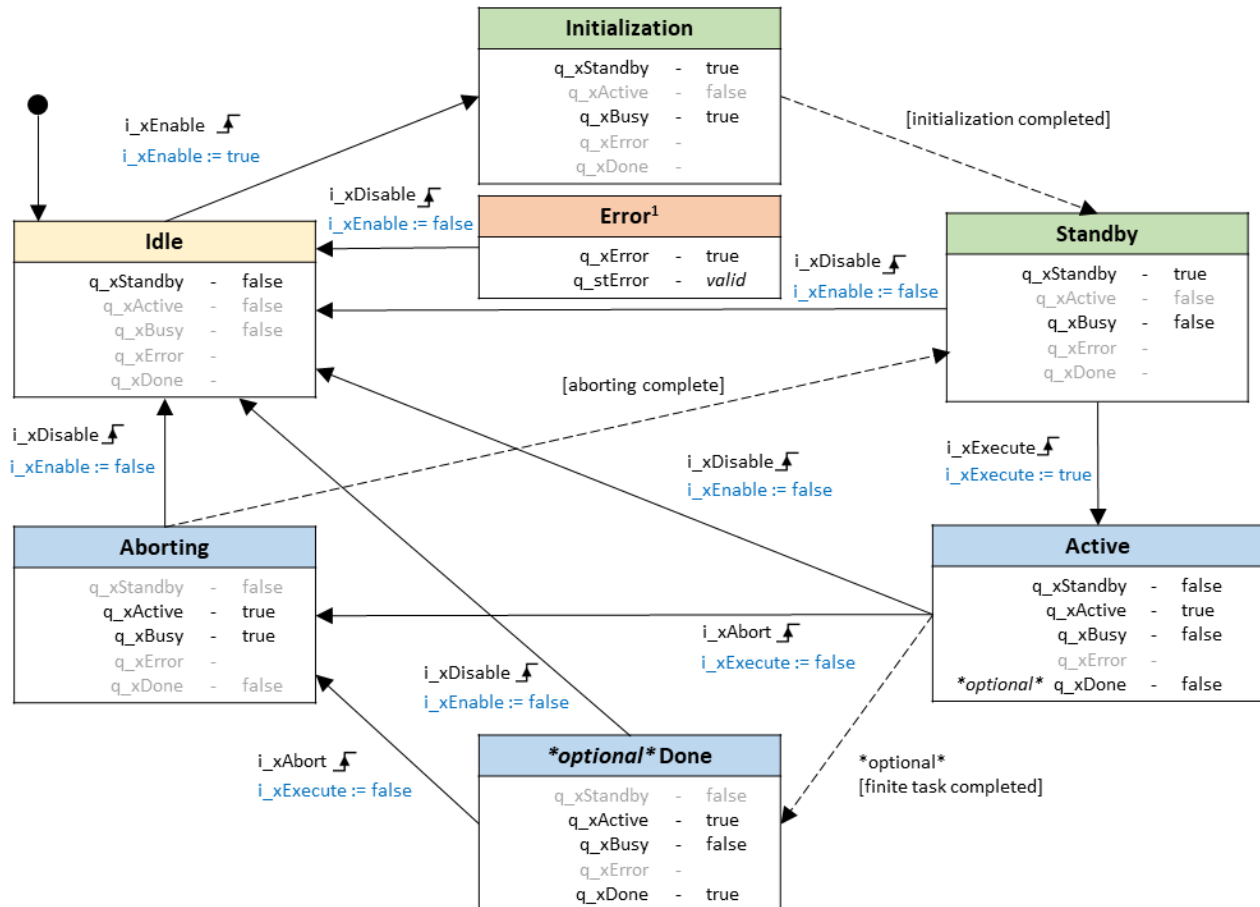
Output	Description
q_xStandby	If true , the FB is either initializing, which may take multiple cycles (output q_xBusy is true), or already initialized and ready to be started (output q_xBusy is false)
q_xActive	If true , the FB is active. The FB is currently performing the desired task (q_xDone and q_xBusy 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.

Structs:

Name	Type	Initial	Comment
ST_ErrorInfo	STRUCT		Contains detailed error information.
xErrorDataValid	BOOL	false	Error information within the struct is only valid if true
strModule	STRING	""	Name of the corresponding function block.
strSource	STRING	""	Name of device or function block where the error occurred first. Can be equal to strModule.
strReason	STRING	""	Error definition
arstrParameter	ARRAY[1..3] OF STRING		Additional error information
dtTimestamp	DATE_AND_TIME		Time and date when error occurred

3.4 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.



3.5 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**.

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.

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`).

3.6 Error information handling

The error information in the struct `ST_ErrorInfo` provides detailed information about an error, such as the source of an error and which software module it caused. It also tells the reason and - if relevant - provides parameter- or variable names in `arstrParameter`. The timestamp indicates when the error was detected.

Structs

Name	Type	Comment
ST_ErrorInfo	STRUCT	Structure with detailed error information
xErrorDataValid	BOOL	True indicates that the provided error information is valid
strModule	STRING(255)	name of the software artefact that has caused the error
strSource	STRING(255)	the source of the error
strReason	STRING(255)	the reason for the error
arstrParameter	ARRAY [1..3] OF STRING(255)	parameter- or variable names involved
dtTimestamp	DT	the error was detected at this time

4 Module parametrization

Parameters of u-remote modules can be read and write by the Codesys application during runtime. The actual implementation of the parameter read or write actions depends on how the modules are connected to the PLC. The modules can be connected either directly to a Weidmueller PLC (via the System Bus) or via a field bus with the respective field bus coupler, e.g., EtherCAT. For each field bus, the implementation might be different.

This chapter explains the common behavior of the module-specific parameter read - and - write function blocks provided in the library package. Please consult the individual FB descriptions for the module-specific properties of these function blocks.

4.1 System Bus

The u-control PLCs have a high-speed serial bus interface to Weidmüller's u-remote remote input / output modules, called *System Bus*. The PLCs and the modules exchange the process image, configuration data and asynchronous information like process alarms and diagnostic alarms via the System Bus.



Parametrization during runtime via the System Bus requires the IoDrvUr20Control library in version 1.0.1.0 or higher. Install the CODESYS Control SL for Weidmueller u-OS package 4.9.0.0 (or higher) provided in the Weidmüller Support Center. It contains the required library and device editor versions.

FB_UR20_<module name>_ParamSysBusRead

These function blocks derive from the FB_LevelControlledFiniteBehavior model. They read the parameter set from a UR20_<module name> u-remote module connected to the System Bus of a u-control with u-OS.

The user enables the function block. The FB checks that `ip_rmiModuleInstance` is a valid module instance and that it references a module of the correct type. Then the FB enters the state **standby**.

The user activates the function block. In state **active**, the function block then reads a parameter byte array from the module instance, translates it into a struct and provides the struct at its output `q_stDeviceParam`. Then the function block changes to **done** state.

If the user wants to read the parameters again, the user deactivates the function block. During the transition back to **standby**, the function block sets `q_stDeviceParam` to default values. The user re-activates the function block, and the function block reads the parameter set, again.

Note: Assign the instance of your UR20-<specific module> in the device tree to `ip_rmiModuleInstance`. Here is an example:

```
fbParamRead(ip_rmiModuleInstance := UR20_2AI_SG_24_DIAG_1);
```

Common Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_rmiModuleInstance	REFERENCE TO IoDrvUR20Control.IoDrvUR20ModuleDiag	module instance to read the parameters from

Common Outputs

Name	Type	Comment
q_xStandby	BOOL	waiting for activation
q_xActive	BOOL	function block activated
q_xBusy	BOOL	function block is activated and doing its supposed task
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_stDeviceParam	ST_<module name>_DeviceParam	parameter set that was read

Common Possible Errors

Possible error messages on FB output q_stError

Error interface (Reason)	Description
invalid module reference	invalid input parameter "ip_rmiModuleInstance"
wrong module TypeName	ip_rmiModuleInstance references the wrong module type
reset error	The sub-FB exposed an error during deactivation.
wrong data size	fbAsyncParamRead reports a wrong parameter set size. Please contact Weidmueller service and provide a detailed error description.
Conversion error	The module's actual parametrization mismatches the FB's known parameter space. Please contact Weidmueller service and provide a detailed error description.
Sub-fb error	fbAsyncParamRead reports an error. Refer to IoDrvUR20Control -> Enums -> Error.

FB_UR20_<module name>_ParamSysBusWrite

These function blocks derive from the FB_LevelControlledFiniteBehavior model. They write the parameter set in ip_stDeviceParam to a UR20-<module name> u-remote module connected to the System Bus of a u-control with u-OS.

The user enables the FB. The FB checks that ip_rmiModuleInstance is a valid module instance and that it references a module of the correct type. Then the FB enters the state **standby**. In state **standby**, the FB checks readiness of its communication sub-FB.

The user activates the FB. The FB checks validity of the parameter set in ip_stDeviceParam, then it shadow-copies internally the parameter set in ip_stDeviceParam and enters the state **active**. In the state **active**, the FB translates the shadow-copied parameter set into a byte array and sends that to the <module name> module via a communication sub-FB. If any checks fail or

the sub-FB reports an error, then the FB enters its state **error**. The FB leaves the state **error** after the user deactivates and disables the FB.

To send another parameter set, the user deactivates the FB. The FB returns to its state **standby**. The user applies a new parameter set to the input `ip_stDeviceParam`, then the user activates the FB again. The FB then translates and sends the new parameter set to the module.

Note: Assign the instance of your UR20-<specific module> in the device tree to `ip_rmiModuleInstance`. Here is an example:

```
fbParamWrite(ip_rmiModuleInstance := UR20_2AI_SG_24_DIAG_1);
```

Common Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active
<code>ip_rmiModuleInstance</code>	REFERENCE TO <code>IoDrvUR20Control.IoDrvUR20ModuleDiag</code>	module instance to read the parameters from
<code>ip_stDeviceParam</code>	ST_UR20_<module name>_DeviceParam	parameter set to be written

Common 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_xError</code>	BOOL	function block is in error state
<code>q_stError</code>	ST_ErrorInfo	detailed error information
<code>q_xDone</code>	BOOL	execution has come to its supposed end

Common Possible Errors

Possible error messages on FB output `q_stError`

Error interface (Reason)	Description
invalid module reference	Invalid input parameter "ip_rmiModuleInstance".
wrong module TypeName	ip_rmiModuleInstance references the wrong module type.
reset error	The sub-FB exposed an error during deactivation.
invalid process value	<i>Please find the module specific parameter errors in the individual FB's descriptions.</i>
sub-fb error	fbAsyncParamWrite reports an error. Refer to IoDrvUR20Control -> Enums -> Error.

4.2 EtherCAT

For u-remote modules connected via EtherCAT and a respective EtherCAT fieldbus coupler, Weidmueller's CODESYS library package provides some module-specific parameter read- and write function blocks for parameterization.

FB_<module name>_ParamEcCanRead

This function block reads the parameters from a UR20 module connected to an EtherCAT or CanOpen fieldbus coupler.

After the user activates the function block, the function block enters its state **active** and begins reading the module parameters. During reading, the function block checks for errors. After all parameters are read, the function block changes to state **done**.

During the transition back to **standby**, the function block replaces the output values in q_stDeviceParam with default values.

Note that the behavior of the function block bases on the FB_LevelControlledFiniteBehavior Model. Please consult the behavior model library description for more information.

Common Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active

Common Outputs

Name	Type	Comment
q_xStandby	BOOL	waiting for activation
q_xActive	BOOL	function block activated
q_xBusy	BOOL	function block is activated and doing its supposed task
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_stDeviceParam	ST_UR20_<module name>_DeviceParam	parameter set that was read

FB_<module name>_ParamEcCanWrite

The user activates the function block. In state **active**, the function block writes parameters to a UR20 module connected to an EtherCAT or CanOpen fieldbus coupler.

After the function block has written all parameters, it changes to state **done**.

Note that the behavior of the function block bases on the FB_LevelControlledFiniteBehavior model. For more information, please consult the behavior model description.

Common Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active

Common 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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

Common Possible Errors

Possible error messages on FB output q_stError

Error interface (Reason)	Description
Invalid parameter "ip_stDevice"	The parameter requires mandatory an input
Invalid process "ip_stDevice"	Input device could not be found during transition to state active.

5 Diagnosis alarm- and process alarm handling

Several u-remote modules feature parameter-switchable monitoring of the sensors connected to the modules' channels. If a module channel is parametrized to do sensor monitoring, it will supervise the sensor connection and / or sensor power supply. If a module detects a line break or an overload, it sends a *diagnosis alarm message* asynchronously.

Also, several u-remote modules feature parameter-switchable monitoring of the process values measured by the modules' channels. If a module channel is parametrized to do process value monitoring, it will compare the actual channel value against parametrized thresholds. If the actual channel value exceeds the upper or lower threshold, the module will send a *process alarm message* asynchronously.

The actual way of reading diagnosis or alarm data depends on how the modules are connected to the PLC. The modules can be connected either directly to a Weidmueller PLC (via the System Bus) or via a field bus with the respective field bus coupler, e.g., EtherCAT. For each field bus, the implementation might be different.



Diagnosis alarm- and process alarm handling via the System Bus requires the IoDrvUr20Control library in version 1.0.1.0. Install the CODESYS Control SL for Weidmueller u-OS package 4.9.0.0 (or higher) provided in the Weidmüller Support Center, it contains the required library and device editor versions.

5.1 Diagnostic alarm messages

A diagnostic alarm message consists of 47 bytes that transport information on the type of error, the module type, the channel type, channel-specific error information and a 32-bit time stamp.

The module-specific libraries contain conversion FCs that translate the 47 bytes into module-specific struct DUTs to provide easier interpretation of the diagnostic alarm messages.



The u-remote handbook provides a description of these diagnostic alarm messages for each module type. It is possible to query diagnostic alarm messages from all module types; However, only modules whose names end with “-DIAG” will raise diagnostic alarms and provide meaningful diagnostic alarm messages.

Diagnostic alarm message transfer via EtherCAT

The EtherCAT fieldbus coupler receives and stores asynchronous diagnostic alarm messages sent by the UR20 modules. This feature needs to be activated in the EtherCAT field bus coupler's configuration in its web interface.

Weidmüller provides a function block FB_UR20_DiagData_Ethercat() in the libWiUr20Diag library. This function block reads the diagnostic data stored in the Ethercat coupler and provides them to the user in an array. The user then checks bytes 2 and 3 of these messages: 0xE7FF indicates a process alarm message; 0xE001 indicates a diagnostic alarm message. The user ignores the bytes 4 .. 23 and finds the diagnostic alarm message in bytes 24 – 70.

Please use the struct DUTs and Function Calls in the module-specific libraries for the translation of the 47 bytes diagnostic alarm messages into a more programmer-friendly format.

Diagnostic alarm message transfer via System Bus

To let your application manage U-Remote diagnosis- and process alarms, you need to enable process alarms and diagnosis alarms globally in the configuration of the U-Control System Bus. To let the U-Remote module produce diagnosis alarm messages and / or process alarm messages, you need to enable the associated features in the module's configuration. These features depend on the module type. Please refer to the U-Remote handbook for the specific configuration of your module type.

The CODESYS System Bus driver library provides some module methods and -attributes for the management of diagnosis alarm messages. These are:

```
<module instance>.Diagnosis
```

This attribute is true when the module instance holds a new diagnosis alarm message.

```
<module instance>.GetDeviceDiagnosis()
```

This method fetches the latest raw 47-byte diagnosis alarm message from the module instance.

```
<module instance>.AckDeviceDiagnosis()
```

This method acknowledges the latest diagnosis alarm message available in the module instance.

```
<module instance>.DiagnosisAcknowledged
```

This attribute is true when there are no un-acknowledged diagnosis alarm messages available in the module instance. Either there are no messages at all, or the application has already acknowledged the latest message.

5.2 Process alarm messages

A process alarm message consists of 2 bytes that transport information on the process alarm(s) themselves and a 16-bit timestamp. This timestamp represents the two least significant bytes of the module's internal 32-bit timer.

For some analogue modules, e.g., the UR20-4AI-RTD-DIAG, the first byte holds a boolean flag for each channel that exceeds its parametrized upper limit value and the second byte holds a Boolean flag for each channel that exceeds a parametrized lower limit value. Other modules use a very different mapping of the process alarm message bytes, however.



The u-remote manual provides a description of the process alarm message bytes for each module type. Please note that not all UR20 modules support the process alarm feature.

Process alarm message transfer via EtherCAT

The user sets the parameter *Process Alarm* in the web interface of the field bus coupler to **enabled** and also enables the desired process alarm in the module's parametrization.

The process value exceeds a limit and the module sends a process alarm message to the field bus coupler. The fieldbus coupler enqueues the message in its diagnostic messages ring buffer and sets the *Summarized Module Diagnosis* flag in its coupler status bits. (The field bus coupler sends the coupler status bits together with the process input data.)

The user uses the *FB_UR20_DiagData_Ethercat()* (as described in diagnostic alarm message transfer via EtherCAT, see above,) to collect the messages currently available in the ring buffer.

Then the user checks bytes 2 and 3 of these messages: 0xE7FF is a process alarm message; 0xE001 is a diagnostic alarm message. The user ignores bytes 4 .. 23 and finds the process alarm message in bytes 24 .. 27.

Process alarm message transfer via System Bus

A U-Remote module sends a process alarm message of four bytes length if it detects violation of a process limit value. Please consult the U-Remote handbook for the module-specific meaning of these four bytes. The module-specific CODESYS libraries provided by Weidmueller also provide function calls that convert a raw 4-byte message to a more convenient struct DUT.

The CODESYS System Bus driver library provides some module methods and -attributes for the management of process alarm messages. These are:

```
<module instance>.ProcessAlarm
```

This attribute is true when the module instance holds a new process alarm message.

```
<module instance>.GetDeviceProcessAlarm()
```

This method fetches the latest raw process alarm message from the module instance.

```
<module instance>.AckDeviceProcessAlarm()
```

This method acknowledges the latest process alarm message available in the module instance.

```
<module instance>.ProcessAlarmAcknowledged
```

This attribute is true when there are no un-acknowledged process alarm messages available in the module instance. Either there are no messages at all, or the application has already acknowledged the latest message.

6 libWiUr20ComRs_232_485_422

The UR20-1COM-232-485-422 module features a UART for serial data transmission via either *RS232*, *RS485* or *RS422*.

6.1 FB_UR20_1ComRs_232_485_422_Control

The UR20-1COM-232-485-422 control function block sends the TX data in iq_arbTransmitData via the UR20-1COM-232-485-422 module or it receives serial data via the UR20-1COM-232-485-422 and provides them in iq_arbReceiveData.

The FB_UR20_1ComRs_232_485_422_Control derives from the level controlled finite behavior model. The user application and the function block (FB) interact as follows:

The application provides parametrization and TX data to the FB. Then it sets i_xEnable to **true** to initialize the FB. The FB validates the parametrization and enters its state **standby** if the parameters are valid or it enters its state **error** otherwise.

In state **standby**, the FB monitors the UR20-1COM-232-485-422 module for incoming serial data or a pending RX buffer overflow. If there is incoming serial data, the FB sets its output q_xReceivedNewMessage to true. If the RX buffer has less than 10 free bytes left, the FB sets q_xReceivedBufferNearlyFull to **true**.

The user application can now use the FB and the UR20-1COM-232-485-422 module to *send* or to *receive* serial data.

Receive serial data:

The user application has initialized the FB and the FB is in **standby**. The user application has set i_etReadWriteMode to ET_UR20_1ComRs_232_485_422_ReadWriteMode.ReadFix and waits until the FB signals incoming serial data (indicated by q_xReceivedNewMessage = **true**).

Then the user application sets i_xExecute to true to activate the FB. The FB collects the incoming serial data from the module. While the transfer is ongoing, the FB signals its ongoing activity with q_xBusy = **true**. After the FB has collected i_diTransactionSize bytes of serial data from the UR20-1COM-232-485-422 module, it signals completion of its task with q_xBusy = false and q_xDone = **true**.

Note that any incoming serial data exceeding i_diTransactionSize bytes remain in the UR20-1COM-232-485-422's RX buffer until the user application deactivates and activates the FB again to fetch the next batch of RX data from the UR20-1COM-232-485-422 module.

Send serial data:

The user application can select two TX modes provided by the UR20-1COM-232-485-422 module:

- direct sending: the 1COM module sends the bytes as they arrive in the module
- triggered sending: the 1COM module collects the bytes in its TX buffer and only starts sending on the serial line after the FB has told it to do so.

Choose by setting `ip_stControlParameter.etTxBufferBehavior` to `<...>.DirectSending` or `<...>.TriggeredSending`.

The user application has initialized the FB and the FB is in **standby**. The user application has set `i_etReadWriteMode` to `ET_UR20_1ComRs_232_485_422_ReadWriteMode.Write` and has provided serial data to send in `iq_arbTransmitData`. The user application sets `i_xExecute` to **true** to activate the FB. The FB signals its ongoing activity by setting `q_xBusy` to **true** and transfers `i_diTransactionSize` bytes of serial data to the UR20-1COM-232-485-422 module. It also starts the `ip_stControlParameter.tTransmissionWatchdogTime` transfer timeout.

After the FB has transferred `iq_arbTransmitData` bytes of TX data, it stops the transfer timeout timer.

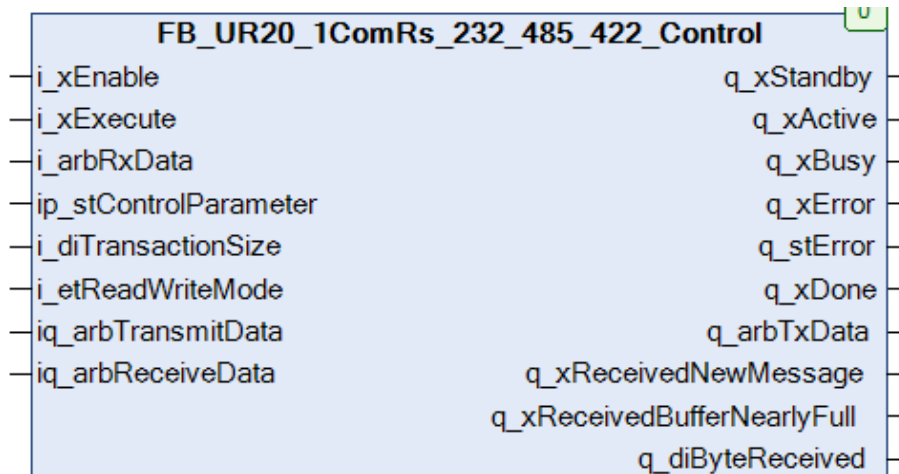
If the user application chose triggered sending, the FB signals the UR20-1COM-232-485-422 module to send the TX data on the serial line.

The FB waits out the `ip_stControlParameter.tWaitTimeEnableTX_HWBUFFER` time, then it sets `q_xBusy` to **false** and `q_xDone` to **true** to signal completion of its activity.

If the transfer timeout timer triggers before the FB has transferred all TX data to the UR20-1COM-232-485-422 module, the FB enters its error state and reports the error on its error interface.



At slow baud rates, it is possible to flood the UR20-1COM-232-485-422 module in direct sending mode: It takes one systembus cycle to transfer 14 bytes of tx data to the UR20-1COM-232-485-422 module. With, e.g., a system bus cycle time of 4 ms, the PLC can transfer 3500 bytes per second to the UR20-1COM-232-485-422 module. So, at any baud rate below 3500 Bd, the UR20-1COM-232-485-422 module's 256 bytes tx buffer will fill and eventually overflow if the user application continuously sends data. If you plan to send in direct mode at slow baud rates, please make sure that the UR20-1COM-232-485-422 module's tx buffer does not overflow.



Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active

Name	Type	Comment
i_arbRxData	ARRAY [0..15] OF USINT	connection to hardware / process data input
ip_stControlParameter	ST_UR20_1ComRs_232_485_422_ControlParam	contains parameter for operating the module
i_diTransactionSize	DINT	amount of bytes/segments to be send or read
i_etReadWriteMode	ET_UR20_1ComRs_232_485_422_ReadWriteMode	defines the type of operation
iq_arbTransmitData	POINTER TO BYTE	input for all data to be send
iq_arbReceiveData	POINTER TO BYTE	output of all the data to be read

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_xError	BOOL	function block in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arbTxData	ARRAY [0..15] OF USINT	connection to hardware / process data output
q_xReceivedNewMessage	BOOL	status from module, new data available
q_xReceivedBufferNearlyFull	BOOL	status from module, only 10 bytes left in buffer
q_diByteReceived	DINT	number of bytes received during read command

Structs

Name	Type	Initial	Comment
ST_UR20_1ComRs_232_485_422_ControlParam	STRUCT		Parameter for operation
etTxBufferBehavior	ET_UR20_1ComRs_232_485_422_TxBufferBehavior		defines the behavior of the TX buffer
etCreateModbusCRC	ET_UR20_1ComRs_232_485_422_CreatModbusCRC		activates and deactivates the local CRC check for modbus RTU
tTransmissionWatchdogTime	TIME	TIME#1s	time to stop ongoing transmission after no response has been received
xTransmissionWatchdogActive	BOOL	TRUE	toggle transmission watchdog on and off
tMemoryFlushTimer	TIME	TIME#100ms	waiting time for flushing the RX and TX buffer
tMemoryFlushTimeOut	TIME	TIME#1s	timeout for flushing the device memory
xMemoryFlushTimerActive	BOOL	TRUE	toggle the memory flush timer

tWaitTimeEnableTX_HWBUFFER	TIME	TIME#100 ms	waiting time after "DisableSend_TX_HWBUFFE R = 0" to enable hardware output puffer
----------------------------	------	----------------	---------------------------------------------------------------------------------------------

Possible Errors

Possible error messages on FB output q_stError

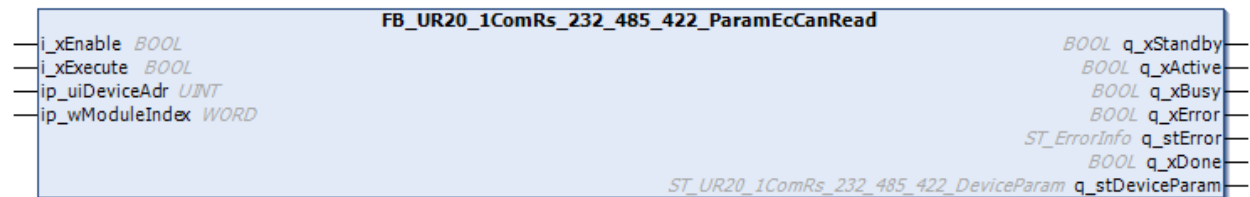
Error interface (Reason)	Description
Communication error	Communication error detected during operation
Frame to long during read	A frame that was received by the module is too long for being read into the output array of the function block
Frame to long during write	A frame that shall be send is to long for the input buffer of the module
Transaction size is larger than the provided TX data	i_diTransactionSize is larger than the actual size of iq_arbTransmitData
Transaction size is too large for TX buffer in buffered write mode	i_diTransactionSize must not be >256 bytes in buffered write mode
Transmission watchdog exceeded	The module did not response within a specific time during a transmission
Invalid parameter	Watchdog time is too short (> 0)
Watchdog for buffer flush expired	The time to flush the buffer was too long.

6.2 FB_UR20_1ComRs_232_485_422_ParamEcCanRead

This function block derives from the FB_LevelControlledFiniteBehavior model. It reads the parameter set from a u-remote module connected to an EtherCAT or CanOpen fieldbus coupler.

The user enables the function block. The function block shadow-copies the field bus coupler address ip_uiDeviceAdr and the u-remote module index ip_wModuleIndex during transition to its state **standby**. The user activates the function block. The function block transitions to its state **active** and reads the parameter set from the module. After all parameters are read, the function block outputs the parameter set on q_stDeviceParam and changes to **done** state. During the transition back to **standby**, the function block writes default values into q_stDeviceParam.

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



Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_uiDeviceAdr	UINT	Address of CANopen or EtherCAT field bus coupler
ip_wModuleIndex	WORD	start index of the u-remote module

Outputs

Name	Type	Comment
q_xStandby	BOOL	waiting for activation
q_xActive	BOOL	function block activated
q_xBusy	BOOL	function block is activated and doing its supposed task
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_stDeviceParam	ST_UR20_1ComRs_232_485_422_DeviceParam	parameter set that was read

Structs

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_DeviceParam	STRUCT	Parameter for operation
etOperatingMode	ET_UR20_1ComRs_232_485_422_OperationMode	defines the used protocol. Default: .disabled
etDataBits	ET_UR20_1ComRs_232_485_422_DataBits	sets the amount of data bits. Default: ._8bit
etBaudRate	ET_UR20_1ComRs_232_485_422_BaudRate	sets the transmission rate. Default: ._9600
etStopBits	ET_UR20_1ComRs_232_485_422_StopBit	defines the amount of stop bits. Default: ._18bit
etParity	ET_UR20_1ComRs_232_485_422_Parity	defines the parity. Default: .None
etFlowControl	ET_UR20_1ComRs_232_485_422_FlowControl	configures the flow control. Default: .None
bXonCharacter	BYTE	defines the ASCII character that is send as XON sign. Default: 17
bXoffCharacter	BYTE	defines the ASCII character that is send as XOFF sign. Default: 19

Possible Errors

Possible error messages on FB output q_stError

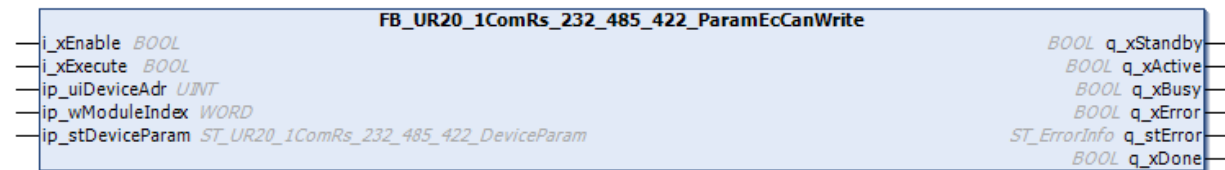
Error interface (Reason)	Description
sub-fb error	fbCopReadSDO reports an error. Refer to EtherCATStack -> Enums -> ETC_CO_ERROR.

6.3 FB_UR20_1ComRs_232_485_422_ParamEcCanWrite

This function block derives from the FB_LevelControlledFiniteBehavior model. It writes a parameter set to a u-remote module connected to an EtherCAT or CanOpen fieldbus coupler. The user enables the function block. The function block shadow-copies the field bus coupler address `ip_uiDeviceAdr`, the u-remote module index `ip_wModuleIndex` and the module parameter set in `ip_stDeviceParam` during transition to its state **standby**. The user activates the function block. The function block transitions to its state **active** and writes the parameter set to the module. After the function block has written the module parameter set, it changes to **done** state. The **error** state can be left by disabling the function block.

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

Note: The module check is done for the whole coupler. The single module is not checked!



Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active
<code>ip_stDeviceParam</code>	ST_UR20_1ComRs_232_485_422_DeviceParam	device parameters
<code>ip_uiDeviceAdr</code>	UINT	Address of CanOpen or EtherCAT field bus coupler
<code>ip_wModuleIndex</code>	WORD	start index of the u-remote module

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_xError</code>	BOOL	function block is in error state
<code>q_stError</code>	ST_ErrorInfo	detailed error information
<code>q_xDone</code>	BOOL	execution has come to its supposed end

Possible Errors

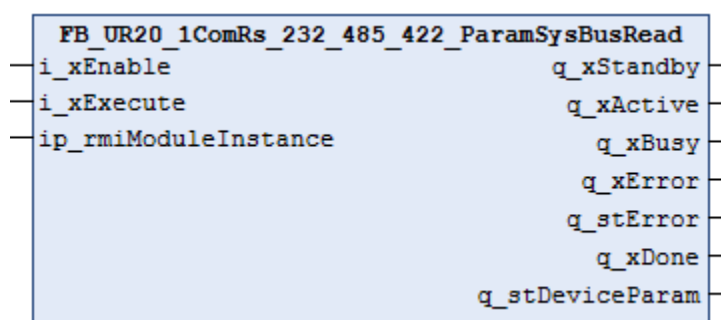
Possible error messages on FB output `q_stError`

Error interface (Reason)	Description
Invalid parameter "ip_stDevice"	The parameter requires mandatory an input
Invalid process "ip_stDevice"	Input device could not be found during transition to state active.
Writing Ethercat Index Parameter	Problem triggered by communication level function block.
Connection lost during action	Connection error while writing the parameters

6.4 FB_UR20_1ComRs_232_485_422_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

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



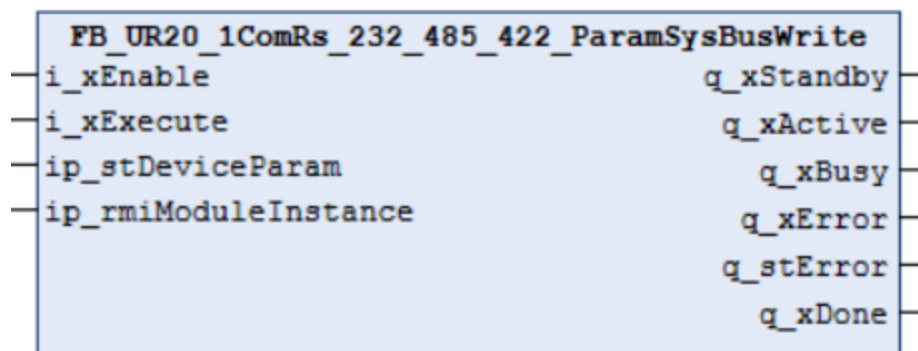
Specific Outputs

Name	Type	Comment
q_stDeviceParam	ST_UR20_1ComRs_232_485_422_DeviceParam	parameter set that was read

3.1 FB_UR20_1ComRs_232_485_422_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

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



Specific Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_1ComRs_232_485_422_DeviceParam	parameter set to be written

Specific Possible Errors

Possible error messages on FB output q_stError

Error interface (Reason)	Description
invalid process value	i_stDeviceParam.etParity must not be "None" when i_stDeviceParam.etDataBits is 7bit.

7 libWlUr20ModbusRTUMaster

7.1 FB_ModbusRTUMaster

The function block FB_ModbusRTUMaster implements a Modbus RTU master that provides communication with a Modbus slave via u-remote module UR20_1COM_232_485_422. This function block derives from the level-controlled finite behavior model. Please consult the Standard Behavior library documentation for further information.

The following paragraphs explain the interaction of the user application (referred to as “user”) with the function block (“FB”).

The user enables the FB and the FB does a check of the `ip_stParameterInputs` parameters during the transition from **idle** to **standby**. If the parameters are not ok, the FB will enter its **error** state instead of **standby**.

The user activates the FB and the FB does a process value check during the transition from **standby** to **active**. If the process values are not ok, the FB will enter its **error** state instead of the state **active**.

In **active** state, the FB performs a ModBus RTU data transfer: Depending on the ModBus function code that the user has provided, the FB either sends data to a ModBus RTU slave device or it reads data from a ModBus RTU slave device. After completion of the transfer, the FB enters its state **active-done**. If the transfer fails, the FB enters the state **error**.

If the user wants to initiate another transfer, the user de-activates the FB, waits for the FB to enter its state **standby** and then sets up the new transfer and activates the FB again.

If the user wants the FB to leave the state **error**, the user disables the FB and enables it again. In state **error**, the FB will provide additional error information on its output `q_stError`.

Send data:

The user sets up the transfer with the following information on the inputs of the FB:

- one of the ModBus RTU Function Codes 5, 6, 8, 15 or 16 on the FB's input `i_enFcCode`.
- the ModBus RTU slave device's address on the FB's input `i_usiSlaveAdr`.
- the start address of the ModBus RTU registers to write on the FB's input `i_uiStartAdr`.
- the amount of data to send on the FB's input `i_uiQuantityData`.
- the data to send in `iq_arxCoilData` for the ModBus RTU Function Codes 5 and 15, or
- the data to send in `iq_arwRegisterData` for the ModBus RTU Function Codes 6, 8 and 16.

Then the user activates the FB and waits until the FB signals completion of the transfer, or its failure.

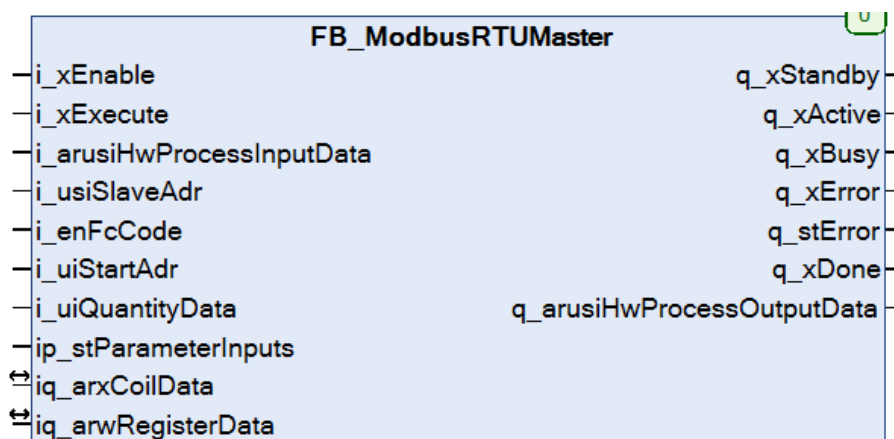
Read data:

The user sets up the transfer with the following information on the inputs of the FB:

- one of the ModBus RTU Function Codes 1, 2, 3 or 4 on the FB's input i_enFcCode.
- the ModBus RTU slave device's address on the FB's input i_usiSlaveAdr.
- the start address of the ModBus RTU registers to read, on the FB's input i_uiStartAdr.
- the amount of data to fetch on the FB's in-/output i_uiQuantityData.

Then the user activates the FB and waits until the FB signals completion of the transfer, or its failure. After successful completion of the transfer, the FB provides the read data on its associated in-/output. This is:

- iq_arxCoilData for the ModBus RTU Function Codes 1 and 2 or
- iq_arwRegisterData for the ModBus RTU Function Codes 3 and 4.



Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
i_arusiHwProcessInputData	ARRAY [0..15] OF USINT	array of process inputs from UR20-1Com Module
i_usiSlaveAdr	USINT	modbus slave address 1-247. Default: 1
i_enFcCode	ET_UR20_ModbusRTU_1ComRs_FunctionCode	modbus function code
i_uiStartAdr	UINT	starting address of modbus register 0-65535. Default: 0
i_uiQuantityData	UINT	quantity of data 1-2000 coils or 1-125 registers. Default: 0
ip_stParameterInputs	ST_UR20_ModbusRTU_1ComRs_ControlParam	struct contains control parameter

Outputs

Name	Type	Comment
q_xStandby	BOOL	waiting for activation

Name	Type	Comment
q_xActive	BOOL	function block is activated
q_xBusy	BOOL	function block is activated and doing its supposed task
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arusiHwProcessOutputData	ARRAY [0..15] OF USINT	array of process outputs to UR20-1Com Module

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

Structs

Name	Type	Comment
ST_UR20_ModbusRTU_1ComRs_ControlParam	STRUCT	Parameter for operation
tReqTimeout	TIME	modbus communication timeout after sending a request. Default: TIME#2s
tHwMemFlushTimer	TIME	waiting period after flushing the TX/RX buffer inside the UR20-1Com Modul, in future not required with new firmware on UR20-1Com Modul. Default: TIME#10ms

Supported Modbus Function Codes

Function Code	Register Type	Explanation
1	Read Coil Status	Reads binary outputs (coils) from a connected slave. The data is stored in Array iq_arxCoilData[1..2000] of BOOL.
2	Read Input Status	Reads binary inputs from a connected slave. The data is stored in Array iq_arxCoilData[1..2000] of BOOL.
3	Read Holding Registers	Reads register-data from a connected slave. The data is stored in Array iq_arwRegisterData[1..125] of WORD.
4	Read Input Registers	Reads input registers from a connected slave. The data is stored in Array iq_arwRegisterData[1..125] of WORD.
5	Write Single Coil	Sends a binary output (Coil) to a connected slave. The value from array iq_arxCoilData[1] is written to the slave.
6	Write Single Register	Sends a single data word to a connected slave. The value from array iq_arwRegisterData[1] is written to the slave.
8	Diagnostics	Sends a diagnostics request with a user defined subfunction code to a connected slave. The subfunction code is passed to the function by the parameter i_uiStartAdr. Additional data can be passed by Array iq_arwRegisterData[1].

Function Code	Register Type	Explanation
15	Write Multiple Coils	Sends several binary outputs (Coils) to a connected slave. The data must be provided in array iq_arxCoilData[1..2000] of BOOL. The maximum number of coils are 0x07B0.
16	Preset Multiple Registers	Sends data to a connected slave. The data must be provided in array arwRegisterData[1..125] of WORD. The maximum number of data are 0x007B.

Possible Errors

Possible error messages on FB output q_stError:

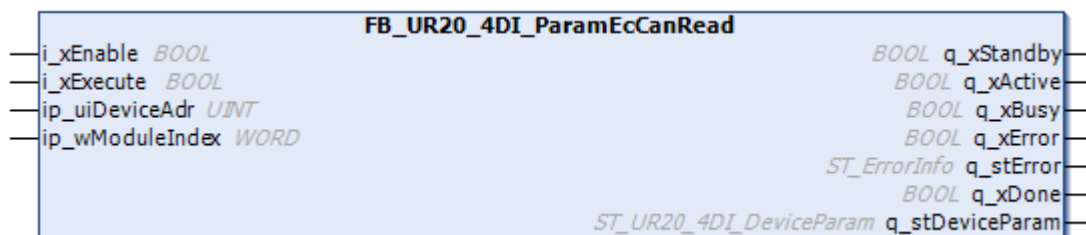
Error interface (Reason)	Description
Invalid parameter	The FB's parameters must be.. a. ip_stParameterInputs.tReqTimeout > T#0S ip_stParameterInputs.tHwMemFlushTimer > T#0S
Invalid process value	Please check the FB's input values and -ranges to be.. a. iq_arxCoilData is a valid reference b. iq_arwRegisterData is a valid reference c. 1 <= i_usiSlaveAdr <= 247 d. 1 <= i_uiQuantityData <= 2000 for FC1 and FC2 e. 1 <= i_uiQuantityData <= 125 for FC3 and FC4 f. 1 <= i_uiQuantityData <= 1968 for FC15 g. 1 <= i_uiQuantityData <= 123 for FC16 h. i_enFcCode is in the supported-list above
Error 1COM driver	Evaluate the forwarded error message.
Timeout 1Com-Driver	Sub-FB doesn't return an answer, please check the MB-Slave communication settings.
Invalid Response	Modbus Slave returns an invalid value
Slave Error Response	Modbus Slave returns an Error code

8 libWlUr20Digital

8.1 FB_UR20_4DI_ParamEcCanRead

The function block reads the parameter set from a u-remote module connected to an EtherCAT or CANopen fieldbus coupler. The function block can be used with the following u-remote Modules: UR20-4DI-P, UR20-4DI-3W, UR20-4DI-N.

The behavior of the function block bases on the FB_LevelControlledFiniteBehavior model.



Inputs

Name	Type	Comment
<i>i_xEnable</i>	BOOL	enables the function block by switching from idle to standby
<i>i_xExecute</i>	BOOL	activates the function block by switching from standby to active
<i>ip_uiDeviceAdr</i>	UINT	Address of the CANopen or EtherCAT field bus coupler
<i>ip_wModuleIndex</i>	WORD	start index of the u-remote module

Outputs

Name	Type	Comment
<i>q_xStandby</i>	BOOL	waiting for activation
<i>q_xActive</i>	BOOL	function block is activated
<i>q_xBusy</i>	BOOL	function block is activated and doing its supposed task
<i>q_xError</i>	BOOL	function block is in error state
<i>q_stError</i>	ST_ErrorInfo	detailed error information
<i>q_xDone</i>	BOOL	execution has come to its supposed end
<i>q_stDeviceParam</i>	ST_UR20_4DI_DeviceParam	parameter set from u-remote module

Structs

Name	Type	Comment
ST_UR20_4DI_DeviceParam	STRUCT	Parameter set for 4DI module
<i>etSwitchOnDelayCh0</i>	ET_UR20_SwitchOnDelayDigital	Switch on delay channel 0
<i>etSwitchOnDelayCh1</i>	ET_UR20_SwitchOnDelayDigital	Switch on delay channel 1
<i>etSwitchOnDelayCh2</i>	ET_UR20_SwitchOnDelayDigital	Switch on delay channel 2

etSwitchOnDelayCh3	ET_UR20_SwitchOnDelayDigital	Switch on delay channel 3
--------------------	------------------------------	---------------------------

Possible Errors

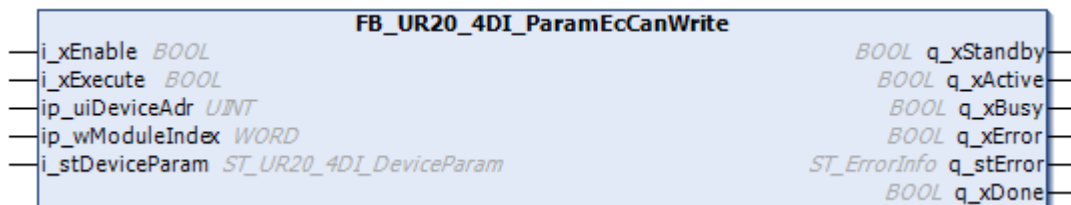
Possible error messages on FB output q_stError

Error interface (Reason)	Description
invalid ip_uiDeviceAdr	This input must have a value assigned to it.
sub-fb error	fbCopReadSDO reports an error. Refer to EtherCATStack -> Enums -> ETC_CO_ERROR.

8.2 FB_UR20_4DI_ParamEcCanWrite

The function block writes the parameter to a u-remote module connected to an EtherCAT or CANopen fieldbus coupler. The function block can be used with the following u-remote modules: UR20-4DI-P, UR20-4DI-3W, UR20-4DI-N.

The behavior of the function block bases on the FB_LevelControlledFiniteBehavior model.



Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active
<code>ip_uiDeviceAdr</code>	UINT	Address of the CANopen or EtherCAT field bus coupler
<code>ip_wModuleIndex</code>	WORD	start index of the u-remote module
<code>i_stDeviceParam</code>	ST_UR20_4DI_DeviceParam	parameter set to be written

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_xError</code>	BOOL	function block is in error state
<code>q_stError</code>	ST_ErrorInfo	detailed error information
<code>q_xDone</code>	BOOL	execution has come to its supposed end

Possible Errors

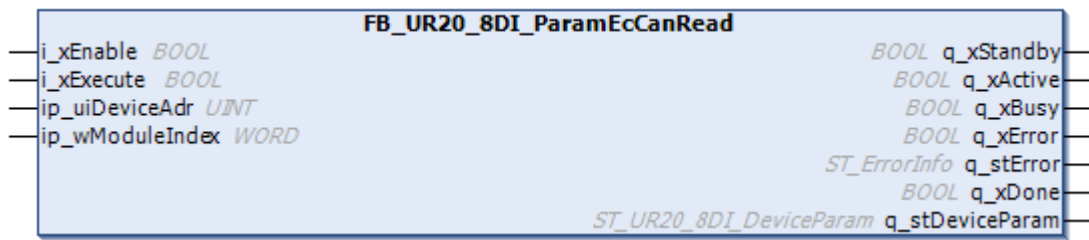
Possible error messages on FB output q_stError

Error interface (Reason)	Description
invalid ip_uiDeviceAdr	This input must have a value assigned to it.
sub-fb error	fbCopWriteSDO reports an error. Refer to EtherCATStack -> Enums -> ETC_CO_ERROR.

8.3 FB_UR20_8DI_ParamEcCanRead

The function block reads the parameters from a u-remote module connected to an EtherCAT or CANopen fieldbus coupler. The function block can be used with the following u-remote modules: UR20_8DI_P_2W, UR20_8DI_P_3W, UR20_8DI_P_3W_HD, UR20_8DI_N_3W, UR20_8DI_ISO_2W.

The behavior of the function block bases on the FB_LevelControlledFiniteBehavior model.



Inputs

Name	Type	Comment
<i>i_xEnable</i>	BOOL	enables the function block by switching from idle to standby
<i>i_xExecute</i>	BOOL	activates the function block by switching from standby to active
<i>ip_uiDeviceAdr</i>	UINT	Address of the CANopen or EtherCAT field bus coupler
<i>ip_wModuleIndex</i>	WORD	start index of the u-remote module

Outputs

Name	Type	Comment
<i>q_xStandby</i>	BOOL	waiting for activation
<i>q_xActive</i>	BOOL	function block is activated
<i>q_xBusy</i>	BOOL	function block is activated and doing its supposed task
<i>q_xError</i>	BOOL	function block is in error state
<i>q_stError</i>	ST_ErrorInfo	detailed error information
<i>q_xDone</i>	BOOL	execution has come to its supposed end
<i>q_stDeviceParam</i>	ST_UR20_8DI_DeviceParam	parameter set that was read

Possible Errors

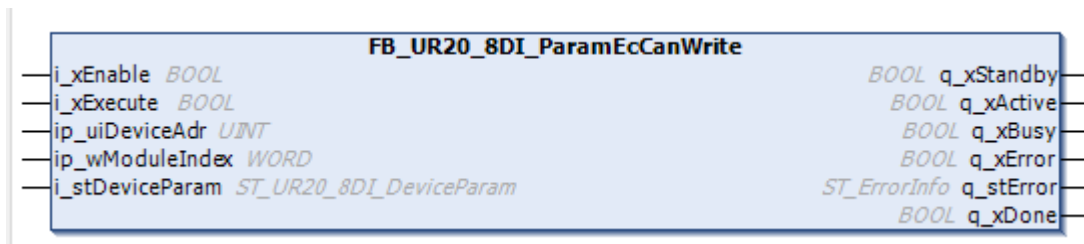
Possible error messages on FB output q_stError

Error interface (Reason)	Description
invalid ip_uiDeviceAdr	This input must have a value assigned to it.
sub-fb error	fbCopReadSDO reports an error. Refer to EtherCATStack -> Enums -> ETC_CO_ERROR.

8.4 FB_UR20_8DI_ParamEcCanWrite

The function block writes the parameter to a u-remote module connected to an EtherCAT or CANopen fieldbus coupler. The function block can be used with the following u-remote modules: UR20_8DI_P_2W, UR20_8DI_P_3W, UR20_8DI_P_3W_HD, UR20_8DI_N_3W, UR20_8DI_ISO_2W.

The behavior of the function block bases on the FB_LevelControlledFiniteBehavior Model.



Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_uiDeviceAdr	UINT	Address of the CANopen or EtherCAT field bus coupler
ip_wModuleIndex	WORD	start index of the u-remote module
i_stDeviceParam	ST_UR20_8DI_DeviceParam	parameter set to be written

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

Possible Errors

Possible error messages on FB output q_stError

Error interface (Reason)	Description
invalid ip_uiDeviceAdr	This input must have a value assigned to it.
sub-fb error	fbCopWriteSDO reports an error. Refer to EtherCATStack -> Enums -> ETC_CO_ERROR.

9 libWiUr20ComRs_232_485_422_V2

This library provides hardware interfaces in the form of function blocks. The implemented interfaces enable the user to control the UR20-1COM-232-485-422 V2 module.

9.1 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.

The plc application on the one hand and the function block on the other hand exchange serial data via two arrays with variable size (iq_arbTransmitData, iq_arbReceiveData). Please declare the array variables you connect here with sufficient size for the size of data you expect to transfer between the UR20-1Com-RS-232-485-422-V2 module and your serial device.

The function block expects and provides the hardware process data in two fixed size arrays: i_arbProcessInputData and q_arbProcessDataOutputData.

The input i_etSendReceiveMode : ET_UR20_1ComRs_232_485_422_V2_SerialOpMode configures the following tasks for serial data communication:

1. Transmit: The FB_UR20_1ComRs_232_485_422_V2_Serial sends a defined amount (i_uiTransactionSize) of data bytes (iq_arbTransmitData) to a connected serial device.
2. ReceiveFixedCycles: The FB_UR20_1ComRs_232_485_422_V2_Serial reads a defined number (i_uiTransactionSize) of cycles. Every cycle, a maximum of 16 Bytes or one telegram can be read.
3. ReceiveTelegrams: The FB_UR20_1ComRs_232_485_422_V2_Serial reads one telegram.
4. ReceiveBuffer: The FB_UR20_1ComRs_232_485_422_V2_Serial reads all data available in the receive buffer of the module.

Once the user sets i_xEnable to true, the function block enters its **standby** state and clears the internal RX and TX buffer of the module.

After the user transitions the function block to the state **active** by also setting i_xExecute, the function block freezes the selected task and performs it until the transaction defined by the selected task is complete. The function block indicates completion of the transaction by setting its output q_xDone to **true**.

The function block's output q_xRxBufferNotEmpty indicates whether new data is available inside the receive buffer of the module.

While the function block is reading data, q_uiBytesRead shows the amount of read bytes.

The telegram configuration is done in the hardware configuration of the module. Please refer to the u-remote manual for further details.

FB_UR20_1ComRs_232_485_422_V2_Serial		
i_xEnable	BOOL	BOOL q_xStandby
i_xExecute	BOOL	BOOL q_xActive
i_arbProcessInputData	ARRAY[0..18] OF BYTE	BOOL q_xBusy
i_etSendReceiveMode	ET_UR20_1ComRs_232_485_422_V2_SerialOpMode	BOOL q_xError
i_uiTransactionSize	UINT	ST_ErrorInfo q_stError
iq_arbTransmitData	ARRAY[*] OF BYTE	BOOL q_xDone
iq_arbReceiveData	ARRAY[*] OF BYTE	ARRAY[0..18] OF BYTE q_arbProcessDataOutputData
		BOOL q_xRxBufferNotEmpty
		BOOL q_xRxBufferFull
		BOOL q_xTxBufferNotEmpty
		BOOL q_xTxBufferFull
		UINT q_uiBytesRead

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
i_arbProcessInputData	ARRAY [0 .. 18] OF BYTE	input for process data send by the hardware to the plc
i_etSendReceiveMode	ET_UR20_1ComRs_232_485_422_V2_SerialOpMode	user input to select between sending and receiving of data
i_uiTransactionSize	UINT	amount of data that shall be handled in current action. can be segments of bytes; depends on selected operation mode
iq_arbTransmitData	POINTER TO BYTE	array of variable size to provide the data that shall be send
iq_arbReceiveData	POINTER TO BYTE	array of variable size that contains the data received after read

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arbProcessDataOutputData	ARRAY [0 .. 18] OF BYTE	output for process data sent by the plc to the hardware
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 cannot take any more data, data loss possible
q_uiBytesRead	UINT	the amount of bytes read during this transaction

InOut

Name	Type	Comment
iq_arbTransmitData	ARRAY [*] OF BYTE	array of variable size to provide the data that shall be sent
iq_arbReceiveData	ARRAY [*] OF BYTE	array of variable size that contains the data received after read

Possible Errors

The FB may indicate the following error messages on output q_stError:

Error interface (Reason)	Description
Input data conversion	The input data is corrupt and cannot be processed further. Please check if the FB's process input data are correctly mapped to the module's process inputs.
Output data conversion	The output data is corrupt. The error is internal and cannot be fixed by the user. Please contact the Weidmüller service.
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.
Reading procedure to long, watchdog exceeded	No response from the module within timeframe during the reading sequence.
Frame to long during write	The frame that shall be written to the module is longer than the input array. Check the transaction size: you attempt to transmit data that are not within the borders of iq_arbTransmitData.
Writing procedure to long, watchdog exceeded	No response from the module within timeframe during the writing 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 during flushing the receive and transmit buffers.

9.2 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. Every channel has its own message configuration with a modbus server ID, 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. 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 reads the data specified by variables `uiOffsetRead` and `uiLengthRead` in `iq_arstModbusChannels` and stores them in `arwRegisterDataRead` or `arxCoilData` in `iq_arstModbusChannels`. A write data channel writes the data stored in `arwRegisterDataWrite` or `arxCoilData` in `iq_arstModbusChannels` to modbus registers specified by variables `uiOffsetWrite` and `uiLengthWrite` in `iq_arstModbusChannels`. Both read and write data channels have 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 fetch or 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 fetch or send data when the FB_UR20_1Com-Rs_232_485_422_V2_ModbusMaster's user sets the associated `xTriggerTransaction` flag.

After it has successfully parametrized the communication channels, the function block confirms entry into the state **standby** by setting `q_xStandby` and waits for the execute signal `i_xExecute` to start its main operation in its state **active**. 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 write configuration, the FB checks if `iq_arstModbusChannels[<channel number>].xTriggerTransaction` is set. If so, it transfers `arwRegisterDataWrite` 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.

For those channel configuration entries set to read and on-demand, the FB tells the module to send the associated Modbus request if `iq_arstModbusChannels[<channel number>]`

].xTriggerTransaction is set. After the FB has sent the request to read to the module, it clears iq_arstModbusChannels[<channel number>].xTriggerTransaction.

For both read and write channels, the FB checks if the module reports a change of a channel's data or diagnosis byte. If there is a change, the FB fetches the changed data and / or the diagnosis information from the module and copies them into etChannelDiag, arwRegisterDataRead or arxCoilData in iq_arstModbusChannels[<channel number>]. To indicate new data or a new diagnosis, the FB sets xNewDataArrived to true. The user needs to reset xNewDataArrived by himself after processing the data.



Important: the function block will read data of a specific channel from the UR20_1COM_232_485_422_V2 only if the module indicates a channel event for that channel. A channel event means that read data or diagnosis information of this channel has changed. The module will always report an **unprocessed channel event of the channel with the lowest channel number, first**. E.g., if channel read data of channel number x changes faster than the FB can process it, read data or diagnosis information of following channels (number > x) **will not be processed**.

To avoid such behavior, make sure that

- iq_arstModbusChannels starts with all write channels, followed by read channels
- read channels in iq_arstModbusChannels are ordered by their read cycle time (or estimated read cycle time for non-cyclic reading), beginning with the slowest. Alternatively, the read channels can be ordered by the expected change interval of the data that is read, beginning with the longest interval.

If there are both write- and read channels, the FB will alternate between processing a write channel, then a read channel, then an event, then a write channel and so on.

To send *new* cyclic data or any on-demand data, write a new payload to arwRegisterDataWrite or arxCoilData in iq_arstModbusChannels. Set iq_arstModbusChannels[<channel number>].xTriggerTransaction to **true**. The function block will then write the new data to the module after other ongoing read or write operations have been completed. 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 the data after you have set iq_arstModbusChannels[<channel number>].xTriggerTransaction 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.

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



Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
i_arbProcessInputData	ARRAY [0 .. 18] OF BYTE	input for process data send by the hardware to the plc

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arbProcessDataOutputData	ARRAY [0 .. 18] OF BYTE	output for process data send by the plc to the hardware

InOut

Name	Type	Comment
iq_arstModbusChannels	ARRAY [*] OF STRUCT	contains the channel configuration, control bits and transaction data

Structs

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_ModbusMasterChannel	STRUCT	Master channel configuration and data
uiCycleTime	UINT	configured cycle time in ms to trigger channel operation internally in the module
uiSlaveAddress	UINT	address (1-247) of slave device the channel communicates with
etFunctionCode	ET_UR20_1ComRs_232_485_422_V2_FunctionCode	configured function code for this channel
uiOffsetWrite	UINT	register or coil address offset that is used by this channel

uiLengthWrite	UINT	amount of register (1-123) or coil (1-1968) written by the channel
uiOffsetRead	UINT	register or coil address offset that is used by this channel
uiLengthRead	UINT	amount of register (1-125) or coil (1-2000) read by the channel
xTriggerTransaction	BOOL	triggers the reading or writing transaction depending on the module configuration, needs to be set if new data shall be written, even if a cycle time has been defined
xNewDataArrived	BOOL	True indicates that the module has sent changed data or changed etChannelDiag
etChannelDiag	ET_UR20_1ComRs_232_485_422_V2_ChannelDiagnosis	diagnostic data of the channel.
arxCoilData	ARRAY [0 .. 1999] OF BOOL	Holds the transaction data if one of the binary function codes is configured for this channel
arwRegisterDataWrite	ARRAY [0..124] OF WORD	stores the transaction send data if one of the analog function codes is configured for this channel
arwRegisterDataRead	ARRAY [0..124] OF WORD	stores the transaction read data if one of the analog function codes is configured for this channel

Possible Errors

The FB indicates the following error messages on output q_stError:

Error interface (Reason)	Description
input data conversion	The input data is corrupt and cannot be processed further. It should be considered to check the mapped data from the module
output data conversion	The output data is corrupt. 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	Somehow a read functioncode was found during 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.

9.3 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. 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.



Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
i_arbProcessInputData	ARRAY [0 .. 18] OF BYTE	input for process data sent by the hardware to the plc
iq_arstModbusChannels	POINTER TO ST_UR20_1ComRs_232_485_422_V2_ModbusSlaveChannel	contains the channel configuration, control bits and transaction data

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arbProcessDataOutputData	ARRAY [0 .. 18] OF BYTE	output for process data sent by the plc to the hardware

InOut

Name	Type	Comment
iq_arstModbusChannels	ARRAY [*] OF STRUCT	contains the channel configuration, control bits and transaction data

Structs

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_ModbusSlaveChannel	STRUCT	Slave channel configuration and data
uiSlaveAddress	UINT	address (1-247) of created slave channel

uiOffset	UINT	register or coil offset (start register) of this channel
uiLength	UINT	register length of slave channel
xWrite	BOOL	possible to write to the slave channel, Modbus FC1 and FC3
xRead	BOOL	sets the channel into read mode
wRegisters	ARRAY [0..124] OF WORD	registers to be processed
xTransactionRead	BOOL	sets the channel to perform a read transaction on next trigger
xTriggerTransaction	BOOL	triggers a read or write transaction from plc to module if the FB is Active

Possible Errors

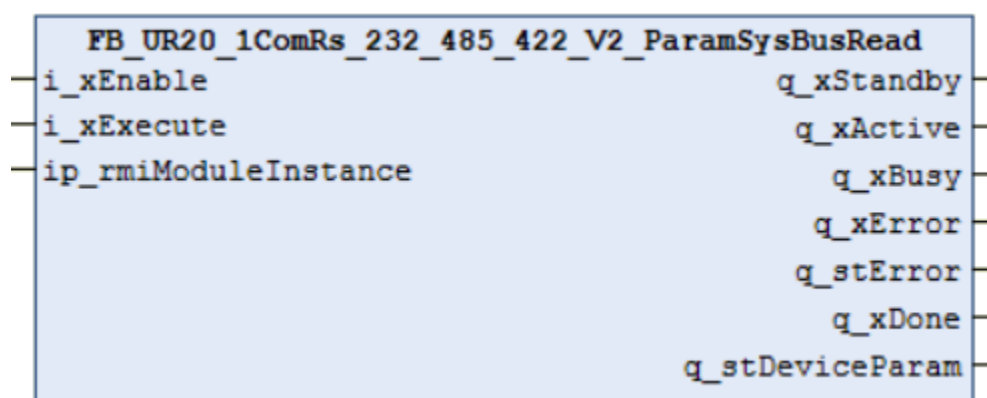
The FB indicates the following error messages on output q_stError:

Error interface (Reason)	Description
Input data conversion	The input data is corrupt and cannot be processed further. Please check the 1COM V2 module's process data mapping.
Output data conversion	The output data is corrupt. The error is internal can not be fixed by the user. Please contact the Weidmüller service.
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.
read/write procedure too long, watchdog timeout 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 too long, watchdog timeout 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 channel number 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.

9.4 FB_UR20_1ComRs_232_485_422_V2_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block shall be used in combination with the following U-Remote modules: UR20-1COM-232-485-422-V2.



Specific Outputs

Name	Type	Comment
q_stDeviceParam	ST_UR20_1ComRs_232_485_422_V2_DeviceParam	parameter set that was read

Structs

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_DeviceParam	STRUCT	This struct represents the UR20-1COM-232-485-422-V2 module's parameter set.
stGeneralSettings	ST_UR20_1ComRs_232_485_422_V2_GeneralSettings	general settings
stFlowControl	ST_UR20_1ComRs_232_485_422_V2_FlowControl	flow control settings
stErrorHandling	ST_UR20_1ComRs_232_485_422_V2_ErrorHandling	error handling settings
stTXSettings	ST_UR20_1ComRs_232_485_422_V2_TXSettings	TX settings
stRXSettings	ST_UR20_1ComRs_232_485_422_V2_RXSettings	RX settings

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_GeneralSettings	STRUCT	This struct represents the General Settings in the module's parameter set.
etOperatingMode	ET_UR20_1ComRs_232_485_422_V2_OperationMode	defines the used protocol

etInterface	ET_UR20_1ComRs_232_485_422_V2_Interface	selects the physical interface
etBaudRate	ET_UR20_1ComRs_232_485_422_V2_BaudRate	sets the transmission rate
etParity	ET_UR20_1ComRs_232_485_422_V2_Parity	selects the parity
etDataBits	ET_UR20_1ComRs_232_485_422_V2_DataBits	sets the amount of data bits
etStopBits	ET_UR20_1ComRs_232_485_422_V2_StopBit	sets the amount of stop bits
etTerminatingResistorRs485_422	ET_UR20_1ComRs_232_485_422_V2_TerminatingResistorRs485_422	termination resistor ON/OFF
etReceivingLinePreset	ET_UR20_1ComRs_232_485_422_V2_ReceivingLinePreset	receiving line preset

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_FlowControl	STRUCT	This struct represents the flow control settings in the UR20-1COM-232-485-422-V2 module's parameter set.
etFlowControl	ET_UR20_1ComRs_232_485_422_V2_FlowControl	flow control selection
bXonCharacter	BYTE	defines the ASCII character that is send as XON sign
bXoffCharacter	BYTE	defines the ASCII character that is send as XOFF sign

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_ErrorHandling	STRUCT	This struct represents the error handling settings in the UR20-1COM-232-485-422-V2 module's parameter set.
etChannelDiagnosis	ET_UR20_1ComRs_232_485_422_V2_ChannelDiagnosisParam	channel diagnosis setting
bErrorPattern	BYTE	error pattern, 0 = Delete incorrect character, 255 = Retain incorrect character

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_TXSettings	STRUCT	This struct represents the TX Settings in the UR20-1COM-232-485-422-V2 module's parameter set.
uiBreakTime	UINT	the break time in bit intervals
uiIdleTime	UINT	the idle time in bit intervals
etSendBreakBeforeStartTelegram	ET_UR20_1ComRs_232_485_422_V2_OnOff	Send break before start telegram
etSendIdleBeforeStartTelegram	ET_UR20_1ComRs_232_485_422_V2_OnOff	Send idle before start telegram
etStartTelegramAfterBreak	ET_UR20_1ComRs_232_485_422_V2_OnOff	Start telegram after break, RS485 operation only

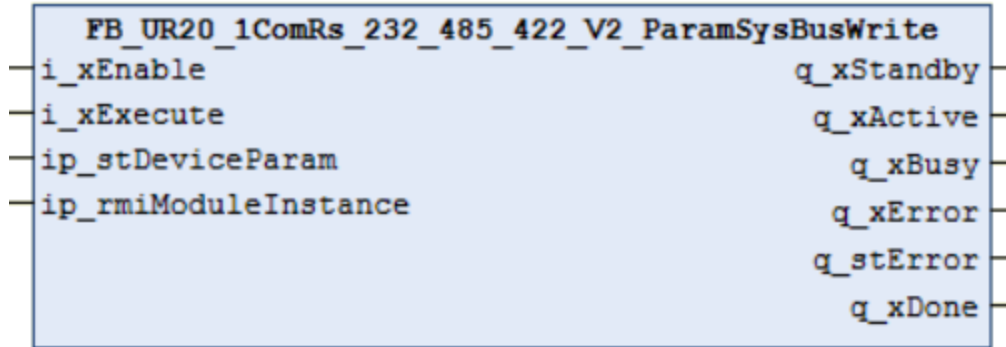
etStartTelegramAfterIdle	ET_UR20_1ComRs_232_485_422_V2_OnOff	Start telegram after idle, RS485 operation only
bNumberOfLeadingChars	BYTE	Number of leading chars, 0..2
bLeadingChar1	BYTE	Leading char 1
bLeadingChar2	BYTE	Leading char 2
bNumberOfAppendedChars	BYTE	Number of appended chars, 0..2
bAppendChar1	BYTE	Append char 1
bAppendChar2	BYTE	Append char 2
uiDMXNumberOfChannels	UINT	DMX: number of channels, 1 ... 513

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_RXSettings	STRUCT	This struct represents the RX Settings in the UR20-1COM-232-485-422-V2 module's parameter set.
uiBreakTime	UINT	the break time in bit intervals
uiIdleTime	UINT	the idle time in bit intervals
etStartTelegramAfterBreak	ET_UR20_1ComRs_232_485_422_V2_OnOff	Start telegram after break
etStartTelegramAfterIdle	ET_UR20_1ComRs_232_485_422_V2_OnOff	Start telegram after idle
etCheckFirstChar	ET_UR20_1ComRs_232_485_422_V2_OnOff	Check first char
bFirstChar	BYTE	first char
etCheckSecondChar	ET_UR20_1ComRs_232_485_422_V2_OnOff	Check second char
bSecondChar	BYTE	second char
uiFixedTelegramLength	UINT	fixed telegram length
etCheckPenultimateChar	ET_UR20_1ComRs_232_485_422_V2_OnOff	Check penultimate char
bPenultimateChar	BYTE	penultimate char
etCheckLastChar	ET_UR20_1ComRs_232_485_422_V2_OnOff	Check last char
bLastChar	BYTE	last char
uiDMXStartChannel	UINT	DMX: start channel
uiDMXLength	UINT	DMX: length

9.5 FB_UR20_1ComRs_232_485_422_V2_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block can be used in combination with the following U-Remote modules: UR20-1COM-232-485-422-V2.



Specific Inputs

Name	Type	Comment
<code>ip_stDeviceParam</code>	<code>ST_UR20_1ComRs_232_485_422_V2_DeviceParam</code>	parameter set to be written

Specific Possible Errors

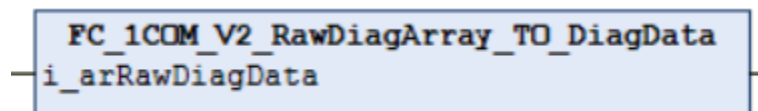
Possible error messages on FB output `q_stError`

Error interface (Reason)	Description
invalid process value	<ul style="list-style-type: none"> a. <code>i_stDeviceParam.etParity</code> must not be "None" when <code>i_stDeviceParam.etDataBits</code> is 7bit. b. <code>i_stDeviceParam.stTXSettings.bNumberOfLeadingChars</code> must be in range 0..2 c. <code>i_stDeviceParam.stTXSettings.bNumberOfAppendedChars</code> must be in range 0..2 d. <code>i_stDeviceParam.stTXSettings.uiDMXNumberOfChannels</code> must be in range 1..513 e. <code>i_stDeviceParam.stRXSettings.uiFixedTelegramLength</code> must be in range 0..2047 f. <code>i_stDeviceParam.stRXSettings.uiDMXLength</code> must be in range 1..513 g. <code>i_stDeviceParam.stRXSettings.uiDMXStartChannel</code> must be in range 0..513

9.6 FC_1COM_V2_RawDiagArray_TO_DiagData

Please consult the Application Note AN0107v1-UC20-u-OS CODESYS Diagnosis- and Process Alarms for an overview over the alarm message transfer.

This function converts a UR20-1COM-232-485-422-V2 module's raw diagnosis alarm message byte array to a struct with a more readable format. Please consult the u-remote manual for details regarding diagnostic alarm messages.



Inputs

Name	Type	Comment
<code>i_arRawDiagData</code>	ARRAY [0..46] OF BYTE	raw diagnostic alarm message from UR20 module

Return

Name	Type	Comment
FC_1COM_V2_RawDiagArray_TO_DiagData	ST_UR20_1ComRs_232_485_422_V2_DiagData	The diagnostic alarm message in a struct.

Structs

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_DiagData	STRUCT	This struct represents the UR20-1COM-RS232-422-485-V2 module-specific part of a diagnostic message. Please find the common part in libWiUr20Diag -> ST_UR20_DiagDataUremote.
stErrorIndicator	ST_UR20_DiagDataErrorIndicator	Byte 0: error indicator
enModuleType	ET_UR20_ModuleType	Byte 1: default value 0x0F
bErrorByte2	BYTE	Byte 2: default value 0
stErrorByte3	ST_UR20_ComModuleDiagDataErrorType	Byte 3: default value 0
enChannelType	ET_UR20_ChannelType	Byte 4: default value 0x70
bNumberOfDiagBits	BYTE	Byte 5: number of diagnostic bit per channel
bNumberOfChannels	BYTE	Byte 6: number of similar channels per module
stErrorChannel	ST_UR20_DiagDataErrorAtChannel	Byte 7: indicates which channel has the error
arbChannelError	ARRAY [0..2] OF BYTE	Byte 8..10: expanded diagnosis
stTxError	ST_UR20_1ComRs_232_485_422_V2_DiagTxError	Byte 11, TX error flags
stRxError	ST_UR20_1ComRs_232_485_422_V2_DiagRxError	Byte 12, RX error flags
stModBusError	ST_UR20_1ComRs_232_485_422_V2_DiagModbusError	byte 13..14, Modbus error flags
stDMXError	ST_UR20_1ComRs_232_485_422_V2_DiagDMXError	Byte 15, DMX error flags
dwTimeStamp	DWORD	Byte 43..46: time stamp µs

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_DiagTxError	STRUCT	This struct represents the TX error flags in the diagnosis alarm message of a 1ComRs_232_485_422_V2 module.
xTxBufferOverflow	BOOL	the TX buffer is overflowed
xLineBreak	BOOL	a line break has occurred
xReserved0	BOOL	this flag is reserved
xReserved1	BOOL	this flag is reserved
xReserved2	BOOL	this flag is reserved
xReserved3	BOOL	this flag is reserved
xReserved4	BOOL	this flag is reserved
xReserved5	BOOL	this flag is reserved

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_DiagModbusError	STRUCT	This struct represents the TX error flags in the diagnosis alarm message of a 1ComRs_232_485_422_V2 module.
xResponseErrorCode01	BOOL	0 Response error code 01
xResponseErrorCode02	BOOL	1 Response error code 02
xResponseErrorCode03	BOOL	2 Response error code 03
xResponseErrorCode04	BOOL	3 Response error code 04
xReserved0	BOOL	4 ... 6 Reserved
xReserved1	BOOL	
xReserved2	BOOL	
xNoResponse	BOOL	7 No response
xMessageTimeout	BOOL	0 Message timeout
xInvalidChannel	BOOL	1 Invalid channel
xInvalidSlaveID	BOOL	2 Invalid Slave ID
xInvalidFunctionCode	BOOL	3 Invalid F-Code
xInvalidRegister	BOOL	4 Invalid register
xInvalidLength	BOOL	5 Invalid length
xInvalidValue	BOOL	6 Invalid value
xInvalidCRC	BOOL	7 Invalid CRC

Name	Type	Comment
ST_UR20_1ComRs_232_485_422_V2_DiagDMXError	STRUCT	This struct represents the DMX error flags in the diagnosis alarm message of a 1ComRs_232_485_422_V2 module.
xMessageTimeout	BOOL	a message timeout has occurred
xReserved0	BOOL	this flag is reserved
xReserved1	BOOL	this flag is reserved
xReserved2	BOOL	this flag is reserved
xReserved3	BOOL	this flag is reserved
xReserved4	BOOL	this flag is reserved
xReserved5	BOOL	this flag is reserved
xReserved6	BOOL	this flag is reserved

9.7 FB_UR20_1ComRs_232_485_422_V2_ModbusRTU_CustomMode

The function block `FB_UR20_1ComRs_232_485_422_V2_ModbusRTU_CustomMode` implements a Modbus RTU master that provides communication with a Modbus slave via u-remote module `UR20_1COM_232_485_422_V2`. This function block derives from the level-controlled finite behavior model. Please consult the Standard Behavior library documentation for further information.

The `FB_UR20_1ComRs_232_485_422_V2_ModbusRTU_CustomMode` implements the Modbus RTU Master inside the plc software of the function block and not within the hardware module directly.

The `FB_UR20_1ComRs_232_485_422_V2_ModbusRTU_CustomMode` use the `FB_UR20_1ComRs_232_485_422_V2_Serial` as a sub function block to write the data to the slave devices.



To use the `FB_UR20_1ComRs_232_485_422_V2_ModbusRTU_CustomMode`, the module must be configured to "Custom" mode (1) and not to "Modbus RTU Master" mode (2).

The following paragraphs explain the interaction of the user application (referred to as "user") with the function block ("FB").

The user enables the FB and the FB performs an initial check of the input parameters during the transition from **idle** to **standby**. If the parameters are not ok, the FB will enter its **error** state instead of **standby**.

The user activates the FB and the FB does a process value check during the transition from **standby** to **active**. If the process values are not ok, the FB will enter its **error** state instead of the state **active**.

In **active** state, the FB performs a Modbus RTU data transfer: Depending on the Modbus function code that the user has provided, the FB either sends data to a Modbus RTU slave device or it reads data from a Modbus RTU slave device. After completion of the transfer, the FB enters its state **active-done**. If the transfer fails, the FB enters the state **error**.

If the user wants to initiate another transfer, the user de-activates the FB, waits for the FB to enter its state **standby** and then sets up the new transfer and activates the FB again.

If the user wants the FB to leave the state **error**, the user disables the FB and enables it again. In state **error**, the FB will provide additional error information on its output `q_stError`.

Send data:

The user sets up the transfer with the following information on the inputs of the FB:

- one of the Modbus RTU Function Codes 5, 6, 8, 15 or 16 on the FB's input `i_enFcCode`.
- the Modbus RTU slave device's address on the FB's input `i_usiSlaveAdr`.
- the start address of the Modbus RTU registers to write on the FB's input `i_uiStartAdr`.
- the amount of data to send on the FB's input `i_uiQuantityData`.
- the data to send in `iq_arxCoilData` for the Modbus RTU Function Codes 5 and 15, or
- the data to send in `iq_arwRegisterData` for the Modbus RTU Function Codes 6, 8 and 16.

Then the user activates the FB and waits until the FB signals completion of the transfer, or its failure.

Read data:

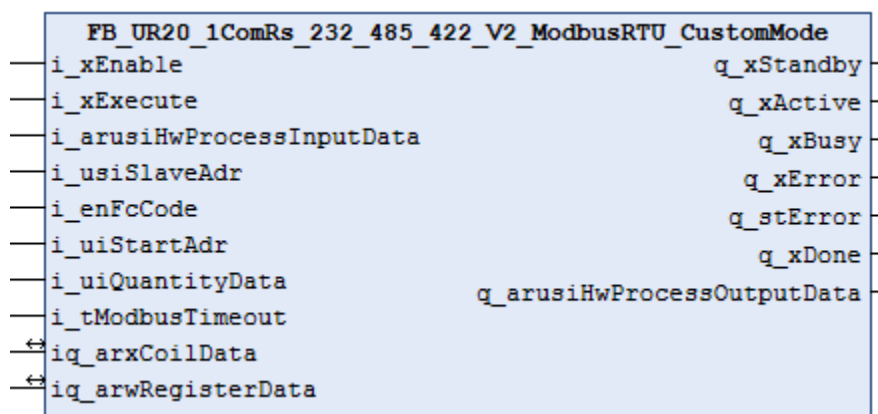
The user sets up the transfer with the following information on the inputs of the FB:

- one of the Modbus RTU Function Codes 1, 2, 3 or 4 on the FB's input `i_enFcCode`.

- the Modbus RTU slave device's address on the FB's input `i_usiSlaveAdr`.
- the start address of the Modbus RTU registers to read, on the FB's input `i_uiStartAdr`.
- the amount of data to fetch on the FB's in-/output `i_uiQuantityData`.

Then the user activates the FB and waits until the FB signals completion of the transfer, or its failure. After successful completion of the transfer, the FB provides the read data on its associated in-/output. This is:

- `iq_arxCoilData` for the Modbus RTU Function Codes 1 and 2 or
- `iq_arwRegisterData` for the Modbus RTU Function Codes 3 and 4.



Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active
<code>i_arusiHwProcessInputData</code>	ARRAY [0..18] OF USINT	array of process inputs from UR20-1Com_V2 Module
<code>i_usiSlaveAdr</code>	USINT	modbus slave address 1-247. Default: 1
<code>i_enFcCode</code>	ET_UR20_1ComRs_232_485_422_V2_FunctionCode	modbus function code
<code>i_uiStartAdr</code>	UINT	starting address of modbus register 0-65535. Default: 0
<code>i_uiQuantityData</code>	UINT	quantity of data 1-2000 coils or 1-125 registers. Default: 0
<code>i_tModbusTimeout</code>	TIME	modbus communication timeout after sending a request

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

Name	Type	Comment
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arusiHwProcess OutputData	ARRAY [0..18] OF USINT	array of process outputs to UR20-1Com Module

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

Supported Modbus Function Codes

Function Code	Register Type	Explanation
1	Read Coil Status	Reads binary outputs (coils) from a connected slave. The data is stored in Array iq_arxCoilData[1..2000] of BOOL.
2	Read Input Status	Reads binary inputs from a connected slave. The data is stored in Array iq_arxCoilData[1..2000] of BOOL.
3	Read Holding Registers	Reads register-data from a connected slave. The data is stored in Array iq_arwRegisterData[1..125] of WORD.
4	Read Input Registers	Reads input registers from a connected slave. The data is stored in Array iq_arwRegisterData[1..125] of WORD.
5	Write Single Coil	Sends a binary output (Coil) to a connected slave. The value from array iq_arxCoilData [1] is written to the slave.
6	Write Single Register	Sends a single data word to a connected slave. The value from array iq_arwRegisterData[1] is written to the slave.
8	Diagnostics	Sends a diagnostics request with a user defined subfunction code to a connected slave. The subfunction code is passed to the function by the parameter i_uiStartAdr. Additional data can be passed by Array iq_arwRegisterData[1].
15	Write Multiple Coils	Sends several binary outputs (Coils) to a connected slave. The data must be provided in array iq_arxCoilData[1..2000] of BOOL. The maximum number of coils are 0x07B0.
16	Preset Multiple Registers	Sends data to a connected slave. The data must be provided in array arwRegisterData[1..125] of WORD. The maximum number of data are 0x007B.

Possible Errors

Possible error messages on FB output q_stError:

Error interface (Reason)	Description
Invalid parameter	The FB's parameters must be.. a. i_tModbusTimeout > T#0S
Invalid process value	Please check the FB's input values and -ranges to be..

Error interface (Reason)	Description
	<ul style="list-style-type: none"> i. iq_arxCoilData is a valid reference j. iq_arwRegisterData is a valid reference k. $1 \leq i_usiSlaveAdr \leq 247$ l. $1 \leq i_uiQuantityData \leq 2000$ for FC1 and FC2 m. $1 \leq i_uiQuantityData \leq 125$ for FC3 and FC4 n. $1 \leq i_uiQuantityData \leq 1968$ for FC15 o. $1 \leq i_uiQuantityData \leq 123$ for FC16 p. i_enFcCode is in the supported-list above
Error 1COM driver	Evaluate the forwarded error message.
Timeout 1Com-Driver	Sub-FB doesn't return an answer, please check the MB-Slave communication settings.
Invalid Response	Modbus Slave returns an invalid value
Slave Error Response	Modbus Slave returns an Error code

10libWiUr203EM_230V_AC

Weidmueller library for parametrization and control of a UR20_3EM_230V_AC energy meter module. Weidmueller strongly recommends that you read the chapter related to the 3EM module in the UR20 manual available on Weidmueller's website before using this library.

10.1 FB_UR20_3EM_Control

Functional Description

This is the UR20-3EM_230V_AC control function block. It transfers measurements from the 3EM module to the user's application and resets the 3EM's energy counters on request by the user. The 3EM module provides up to 56 different measurement values related to the electrical energy metered by the 3EM module. You may freely select which and how many measurement values you want this function block to collect for you:

- determine the number of measurement values and parametrize `ip_st3EMParams.uiNumberOfMeasurementValues` accordingly
- if you use current transformers, parametrize their ratio in `ip_st3EMParams.lCurrentTransformerRatio`
- select the measurement values in `ip_st3EMDeviceParams.areMeasurementsToChannels`
- use one of the `FB_UR20_3EMParam<field bus>Write` function blocks provided in this library to set the UR20_3EM_230V_AC module's device parameters or
- set the UR20_3EM_230V_AC module's device parameters via the CODESYS project.

The 3EM module knows 56 measurement *values* but has only eight *channels* for the transfer of measurement values to the PLC. Therefore, the `FB_UR20_3EM_Control` has two modes of operation:

1. Straight through: The function block uses this mode of operation if you select eight or less measurement values. In this mode of operation, the function block simply converts the values provided via the 3EM's eight channels to SI units and puts them into `q_ar1rMeasurements`.



In this mode, the `FB_UR20_3EM_Control` writes your measurement value choice in `ip_st3EMDeviceParams.areMeasurementsToChannels` to the module *once*, right after you activate the function block.

2. Multiplex: The function block uses this mode of operation if you select more than eight measurement values. In this mode of operation, the function block runs through the following steps cyclic:

- a. parameterize eight measurement values
- b. wait `ip_st3EMParams.tDataTransfer` for the 3EM to send the associated measurement values to the PLC
- c. convert the associated measurement values to SI units and put them into `q_ar1rMeasurements`
- d. repeat the above with the next up to eight measurement values (or go back to the first eight measurement values when no more are left.)



E.g. for a module directly attached to the UC20..OLAC, the above cycle takes four PLC cycles per measurement value plus `ip_st3EMParams.tDataTransfer` plus one PLC cycle. For example, 20 measurement values require 81 PLC cycles plus `ip_st3EMParams.tDataTransfer`. The timing depends on the choice of fieldbus via which the UR20_3EM_230V_AC communicates with the PLC. Since the function block collects the measurement values in sets of eight, your measurement values seen as the *entire set* will not be atomic, anymore. However, they do are atomic over *sets of eight*. Counting from the start of `ip_st3EMDeviceParams.aretMeasurementsToChannels`, each eight measurement values are atomic. Therefore, you may want to arrange the measurement value configuration so that related data (like current, power and voltage of each of the three phases) are in atomic sets.

Note on usage: Weidmueller strictly encapsulates I/O module parametrization in fieldbus specific parameter read and -write function blocks. Therefore, you need to connect the FB_UR20_3EM_Control with a suitable FB_UR20_3EM_ChannelParam<field bus>Write function block provided in this library. This is simple: Declare an instance of the FB_UR20_3EM_ChannelParam<field bus>Write in your POU or FB. Pass `ip_uiDeviceAdr` and, if needed, `ip_wIndex` to that instance. Then connect the instance to FB_UR20_3EM_Control's input `i_fbChannelParamAnyWrite`. Here is a piece of example code for a UR20_3EM_230V_AC attached to the PLC via EtherCAT:

```
fbChannelWrite3EMEtherCAT : FB_UR20_3EM_ChannelParamEcCanWrite;
fbChannelWrite3EMEtherCAT.ip_uiDeviceAdr := 1001; (* tell channel parameter
write FB which device to write to *)
fbChannelWrite3EMEtherCAT.ip_wIndex := 16#8000;
fb3EMControl( <other i./o. omitted here for brevity>, i_fbChannelParamAnyWrite
:= fbChannelWrite3EMEtherCAT); (* execute FB *)
```

The FB_UR20_3EM_Control also provides a conversion of the UR20_3EM_230V_AC module's raw values to SI units in its output variable `q_ar1rMeasurements`. Please observe that the conversion to SI requires a correctly set current transformer ratio in `ip_st3EMParams.lrCurrentTransformerRatio`:

- you use current transformers: their ratio is e.g. 40:1 then set `ip_st3EMParams.lrCurrent-TransformerRatio` to *LREAL#40.0*
- you do not use current transformers: set `ip_st3EMParams.lrCurrentTransformerRatio` to *LREAL#1.0*

The UR20_3EM_230V_AC module has integer 16 bit energy counters. These overflow at 10000 Wh (the actual unit scales with the current transformer ratio). To provide the user with more flexibility, the FB_UR20_3EM_Control extends the energy counters to 32 bit integer FB-internally and provides the needed overflow management: Provided that the user has configured an energy counter measurement value, the FB checks continuously if an overflow has occurred. If so, it calculates new internal extension value(s) and issues an energy counter reset pulse to the

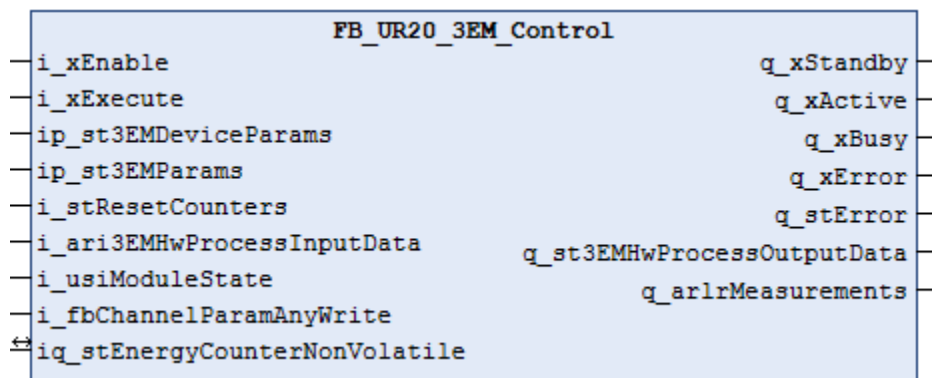
UR20_3EM_230V_AC module. The FB_UR20_3EM_Control calculates the SI values for energy counter measurement values from the sum of its internal energy counter extension and the actual value read from the UR20_3EM_230V_AC module. Because CODESYS provides a choice of approaches for more-or-less non-volatile variable storage on a PLC, the FB_UR20_3EM_Control exposes in its `iq_stEnergyCounterNonVolatile` structure the accumulated overflown energy values. If you desire non-volatile storage, please declare and handle the variable connected to `iq_stEnergyCounterNonVolatile` according to your choice of persistence; See also “Data Persistence” in the CODESYS help.

The FB_UR20_3EM_Control function block also provides you with a means to reset the extended energy counters: Rising edges on the four boolean flags in `i_stResetCounters` reset the associated energy counters in the 3EM module and the FB’s internal energy counter extension. Remember to reset the boolean flags to false after you issued an energy counter reset or the FB’s automatic overflow management will not work as expected.



The FB_UR20_3EM_Control converts UR20_3EM_230V_AC module raw values to SI units for you. For the correct function of this feature, it is necessary that the UR20_3EM_230V_AC module has been parametrized with the same parameters as you input into FB_UR20_3EM_Control via `ip_st3EMDeviceParams` and `ip_st3EMParams`. An easy way to achieve this is to write the same parameters to the module as you provide to FB_UR20_3EM_Control: set up the parameters in the associated variables `ip_st3EMDeviceParams` and `ip_st3EMParams` and connect these variables to both your instances of FB_UR20_3EM_Control and FB_UR20_3EM_Param<field bus>Write. Then execute the FB_UR20_3EM_Param<field bus>Write and wait for its `xDone` signal before you activate FB_UR20_3EM_Control. Please also refer to the documentation of FB_UR20_3EM_Param<Field Bus>Read and FB_UR20_3EM_Param<Field Bus>Write.

The function block FB_UR20_3EM_Control has level-controlled behavior.



Possible Errors

Reason	Description
Invalid parameter	a. ip_st3EMParams.uiNumberOfMeasurementValues is greater than 56. b. ip_st3EMParams.lnCurrentTransformerRatio is zero or <i>negative</i> . c. ip_st3EMParams.tDataTransfer is zero. d. iq_stEnergyCounterNonVolatile is no valid reference. e. i_fbChannelParamAnyWrite is no valid reference.
Invalid device parameter	a. an element of aretMeasurementsToChannels is set to the value reactiveEnergyLeadingCounterL<n> or to the value reactiveEnergyLaggingCounterL<n>, and ip_st3EMDeviceParams.bHarmonicSelect is <> 1. b. an element of aretMeasurementsToChannels beyond the parametrized number of measurements is not set to ET_UR20_3EM_MeasurementValues#deactivated.
Invalid reference	iq_stEnergyCounterNonVolatile became invalid in active state.
UR20_3EM_230V_AC module error	The UR20_3EM module reported an error in active state

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_st3EMDeviceParams	ST_UR20_3EM_DeviceParam	multiplex operation only: FB_UR20_3EM_Control cycles the 3EM module through the measurement values list in aretMeasurementsToChannels
ip_st3EMParams	ST_UR20_3EM_Param	number of measurement values in uiNumberOfMeasurementValues; current transformer ratio in lnCurrentTransformerRatio; multiplex operation only: tDataTransfer time.
i_stResetCounters	ST_UR20_3EM_ResetCounters	rising edge on the four boolean flags in this struct resets the associated energy counters and FB-internal extension structures
i_ari3EMHwProcessInputData	ARRAY [0..7] OF INT	connect to the eight channels in 3EM module -> i/o mapping -> input data
i_usiModuleState	USINT	UR20_3EM 'module state' process input
i_fbChannelParamAnyWrite	REFERENCE TO FB_UR20_3EM_ChannelParamAnyWrite	connect to an instance of FB_ChannelParam<field bus>Write

Outputs

Name	Type	Comment
q_xStandby	BOOL	waiting for activation
q_xActive	BOOL	function block is activated

Name	Type	Comment
q_xBusy	BOOL	function block is activated and doing its supposed task
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_st3EMHwProcessOutputData	ST_UR20_3EM_OutputData	connect to the 3EM's i/o mapping -> output data
q_ar1rMeasurements	ARRAY [0..55] OF LREAL	your SI measurement values in an array of LREAL

InOut

Name	Type	Comment
iq_stEnergyCounterNonVolatile	ST_UR20_3EM_EnergyCounterNonVolatile	energy counters: these structs contain the accumulated overflowed energy values, for optional non-volatile storage.

Structs

Name	Type	Comment
ST_UR20_3EM_DeviceParam	STRUCT	3EM module parameter
etPowerFactorAlarmEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated
lrFrequencyAlarmUpperLimit	LREAL	the upper frequency alert limit in Hz. Default: 65.0
etFrequencyAlarmUpperLimitEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated
lrFrequencyAlarmLowerLimit	LREAL	the lower frequency alert limit in Hz. Default: 45.0
etFrequencyAlarmLowerLimitEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated
lrCurrentImbalanceAlarmLimit	LREAL	the current imbalance limit in the range 0.0 .. 1.0. Default: 1.0
etCurrentImbalanceAlarmEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated
lrCurrentAlarmUpperLimit	LREAL	the upper current alert limit in A. Default: 5.0
etCurrentAlarmUpperLimitEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated
lrCurrentAlarmLowerLimit	LREAL	the lower current alert limit in A. Default: 0.0
etCurrentAlarmLowerLimitEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated
lrVoltageAlarmUpperLimit	LREAL	the upper Voltage alert limit in V. Default: 300.0
etVoltageAlarmUpperLimitEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated
lrVoltageAlarmLowerLimit	LREAL	the lower Voltage alert limit in V. Default: 0.0
etVoltageAlarmLowerLimitEnable	ET_UR20_3EM_AlarmOnOff	Switches this alert on or off. Default: .deactivated

bHarmonicSelect	BYTE	Selects the harmonic several measurements relate to, e.g. harmonicPowerL1. Default: 1
etCurrentRange	ET_UR20_3EM_CurrentRange	Selects current measurement range: 0..1A or 0..5A. Default: . _5A
lrPowerFactorAlarmLowerLimit	LREAL	This is the lower power factor alert limit in the range 0.0 .. 1.0. Default: 0.0
etDiagnosticAlarm	ET_UR20_3EM_AlarmOnOff	Switches all diagnostic alerts on or off. Default: .deactivated
aretMeasurementsToChannels	ARRAY [0..55] OF ET_UR20_3EM_MeasurementValues	Maps measurement values on measurement channels. See the FB_UR20_3EM_Control documentation, too. Default: [56(.disabled)]
ST_UR20_3EM_Param	STRUCT	3EM FB parameters
uiNumberOfMeasurementValues	UINT	the number of measurement values you want FB_UR20_3EM_Control to fetch from the UR20_3EM_230V_AC module for you. Default: 8
tDataTransfer	TIME	multiplex operation mode: FB_UR20_3EM_Control waits for tDataTransfer after a reparametrization of the module's channels until it reads the values. Default: TIME#20ms
lrCurrentTransformerRatio	LREAL	lrCurrentTransformerRatio : 1 is the ratio of the current transformers used with the UR20_3EM_230V_AC module. Set to 1 if you do not use current transformers. Default: 1.0

10.2 UR20_3EM_ChannelParamEcCanWrite

Functional Description

This function block writes a set of eight measurement channel device parameters to an UR20_3EM_230V_AC module attached to an UR20 EtherCAT fieldbus coupler connected to the PLC. Please also read the documentation of FB_UR20_3EM_Control regarding usage of this FB.

A note regarding the input ip_uiDeviceAdr:

To find the required value in your project, open the EtherCAT coupler device in the devices tree. Then open the tab 'General' and use the value shown as 'EtherCAT address'.

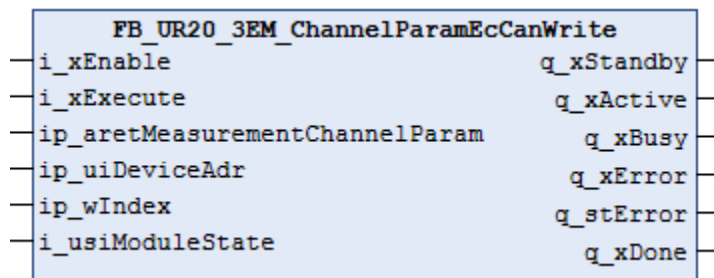
A note regarding the input ip_wIndex:

To find the required value in your project, open the desired UR20_3EM_230V_AC module in your devices tree. In the tab 'Startup Parameters', find the column labelled 'Index:Subindex'. Use the value shown for 'Index'.

The function block can be used in combination with the following u-remote modules: UR20_3EM_230V_AC.



The module check is done for the whole coupler. The single module is not checked!



Possible Errors:

Reason	Description
Invalid parameter	ip_uiDeviceAdr must not be zero
Invalid process value	Invalid value in ip_aretMeasurementChannelParam[] during standby-exit
UR20_3EM_230V_AC module error	The UR20_3EM module reported an error in active state
EtherCAT parameter write error	A parameter write access to the UR20_3EM_230V_AC module has failed

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_aretMeasurementChannelParam	ARRAY [0..7] OF <u>ET_UR20_3EM_MeasurementValues</u>	maps measurement values on measurement channels.
ip_uiDeviceAdr	UINT	Address of CANopen or EtherCAT device [coupler]
ip_wIndex	WORD	start address of the module
i_usiModuleState	USINT	UR20_3EM 'module state' process input

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
i_xExecute	BOOL	activates the function block by switching from standby to active
q_xDone	BOOL	execution has come to its supposed end

10.3 FB_UR20_3EM_ParamEcCanRead

Functional Description

This function block reads all parameters from an UR20_3EM_230V_AC module attached to a PLC via an UR20 EtherCAT fieldbus coupler.

A note regarding the input `ip_uiDeviceAdr`:

To find the required value in your project, open the EtherCAT coupler device in the devices tree. Then open the tab 'General' and use the value shown as 'EtherCAT address'.

A note regarding the input `ip_wIndex`:

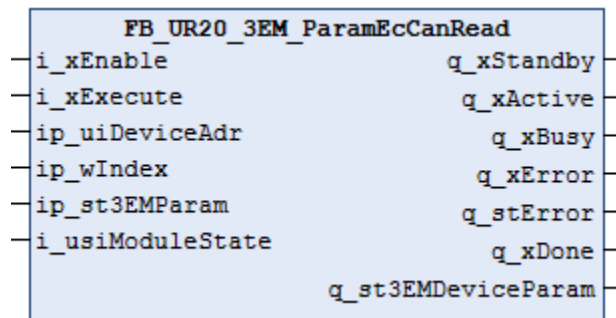
To find the required value in your project, open the desired UR20_3EM_230V_AC module in your devices tree. In the tab 'Startup Parameters', find the column labelled 'Index:Subindex'. Use the value shown for 'Index'.

The function block can be used in combination with the following u-remote module: UR20_3EM_230V_AC.



The module check is done for the whole coupler. The single module is not checked!

FB_UR20_3EM_ParamEcCanRead has level controlled finite behavior.



Possible Errors

Reason	Description
Invalid parameter	a. <code>ip_uiDeviceAdr</code> must not be zero b. <code>ip_st3EMParams.lCurrentTransformerRatio</code> must not be zero or negative.
UR20_3EM_230V_AC module error	The UR20_3EM module reported an error in active state
EtherCAT parameter read error	A parameter read access to the UR20_3EM_230V_AC module has failed

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_uiDeviceAdr	UINT	Address of CANopen or EtherCAT device [coupler]
ip_wIndex	WORD	start address of the module
Ip_st3EMParam	ST_UR20_3EM_Param	The FB's parameters.
i_usiModuleState	USINT	UR20_3EM 'module state' process input

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_st3EMDeviceParam	ST_UR20_3EM_DeviceParam	parameter set that was read from the 3EM module

10.4 FB_UR20_3EM_ParamEcCanWrite

Functional Description

This function block writes a set of parameters to an UR20_3EM_230V_AC module attached to an UR20 EtherCAT fieldbus coupler connected to the PLC.

A note regarding the input ip_uiDeviceAdr:

To find the required value in your project, open the EtherCAT coupler device in the devices tree. Then open the tab 'General' and use the value shown as 'EtherCAT address'.

A note regarding the input ip_wIndex:

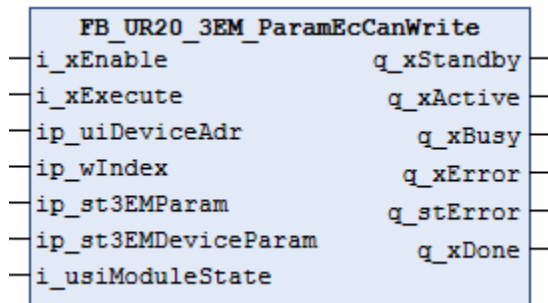
To find the required value in your project, open the desired UR20_3EM_230V_AC module in your devices tree. In the tab 'Startup Parameters', find the column labelled 'Index:Subindex'. Use the value shown for 'Index'.

The function block can be used in combination with the following u-remote module: UR20_3EM_230V_AC.



The module check is done for the whole coupler. The single module is not checked!

FB_UR20_3EM_ParamEcCanWrite has level-controlled finite behavior.



Possible Errors

Reason	Description
Invalid parameter	a. ip_uiDeviceAdr must not be zero b. ip_st3EMParams.lnCurrentTransformerRatio is zero or negative
invalid process value	a. bHarmonicSelect > 31 OR bHarmonicSelect < 1 b. lnLowerVoltageLimit > 300.0 or < 0 c. lnUpperVoltageLimit > 300.0 or < 0 d. lnLowerCurrentLimit > 5.0 or < 0 (with 5A current range and lnTransformerRatio := 1) e. lnUpperCurrentLimit > 5.0 or < 0 (with 5A current range and lnTransformerRatio := 1) f. lnCurrentImbalanceLimit > 1.0 or < 0 g. lnFrequencyAlarmLowerLimit < 45 OR lnFrequencyAlarmLowerLimit > 65 h. lnFrequencyAlarmUpperLimit < 45 OR lnFrequencyAlarmUpperLimit > 65 i. lnPowerFactorAlarmLowerLimit > 1.0 or < 0 j. check if no element of aretMeasurementsToChannels is set to the value reactiveEnergyLeadingCounterL<n> or to the value reactiveEnergyLaggingCounterL<n>, while ip_st3EMDeviceParams.bHarmonicSelect is <> 1
UR20_3EM_230V_AC module error	The UR20_3EM module reported an error in active state
EtherCAT parameter write error	A parameter write access to the UR20_3EM_230V_AC module has failed

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_uiDeviceAdr	UINT	Address of CANopen or EtherCAT device [coupler]
ip_wIndex	WORD	start address of the module
ip_st3EMParam	ST_UR20_3EM_Param	the FB's parameters

Name	Type	Comment
ip_st3EMDeviceParam	ST_UR20_3EM_DeviceParam	parameter set to be written
i_usiModuleState	USINT	UR20_3EM 'module state' process input

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

11 libWiUr20Diag

Weidmueller Interface library for reading diagnostic data of a u-remote module. Weidmueller Interface strongly recommends reading the UR20 manual related to the deployed module available on Weidmueller website before using this library.

Each u-remote module can generate diagnostic data, e.g., in the event of a channel error. The diagnostic data is always an array of 47 bytes.

11.1 FB_UR20_Uremote_Diagdata_Ethercat

Functional Description

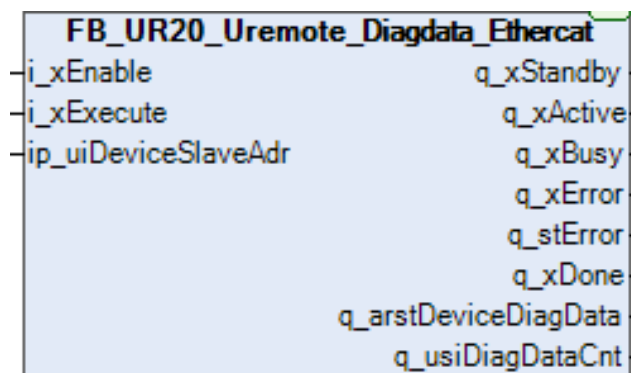
Each u-remote module can generate diagnostic data, e.g. in the event of a channel error. The diagnostic data is always an array of 0..46 bytes. Please take a look at the u-remote manual for further information.

This function block reads the diagnostic data from the u-remote modules connected to an EtherCAT coupler.

The function block reads all available messages at once. The output array element `q_arstDeviceDiagData[0]` shows the newest diagnostic message.

The diagnostic functionality must be activated inside the EtherCAT coupler. Without activation the function block returns an error message.

After all messages are loaded, the function block indicates completion of its finite task by setting `q_xDone` to **true**.



Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active
<code>ip_uiDeviceSlaveAdr</code>	UINT	device slave address from Ethercat coupler

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arstDeviceDiagData	ARRAY [0..20] OF ST_UR20_DiagDataEcUremote	output array contains the diagnostic data
q_usiDiagDataCnt	USINT	counter how many diagnostic data messages are available

Structs

Name	Type	Comment
ST_UR20_DiagDataUremote	STRUCT	Slave channel configuration and data
stErrorIndicator	ST_UR20_DiagDataErrorIndicator	Byte 0 Error Indicator
enModuleType	ET_UR20_ModuleType	Byte 1. Default value: 0x0F
bErrorByte2	BYTE	Byte 2. Default value: 0
stErrorByte3	ST_UR20_DiagDataErrorType	Byte 3. Default value: 0
enChannelType	ET_UR20_ChannelType	Byte 4. Default value: 0x70
bNumberOfDiagBits	BYTE	Byte 5, number of diagnostic bit per channel
bNumberOfChannels	BYTE	Byte 6, number of similar channels per module
stErrorChannel	ST_UR20_DiagDataErrorAtChannel	Byte 7 indicates which channel has the error
arbChannelError	ARRAY [0..3] OF BYTE	BYTE 8..10 expanded diagnosis
arstDiagErrorChannel	ARRAY [0..31] OF ST_UR20_DiagDataErrorAtChannel	Byte 11..42 Error Channel 0..Error Channel 31, diagnostic data
dwTimeStamp	DWORD	BYTE 43..46 time stamp µs
stErrorMessage	ST_UR20_DiagDataText	diag info string format

Possible errors

Error interface (Reason)	Description
Invalid param "ip_uiDeviceSlaveAdr"	The parameter requires mandatory an input
Invalid process "ip_uiDeviceSlaveAdr"	Input device could not be found during transition to state active.
Sub-FB returns an error	Please check the error parameter output for more details

12libWiUr204COM_IOLINK

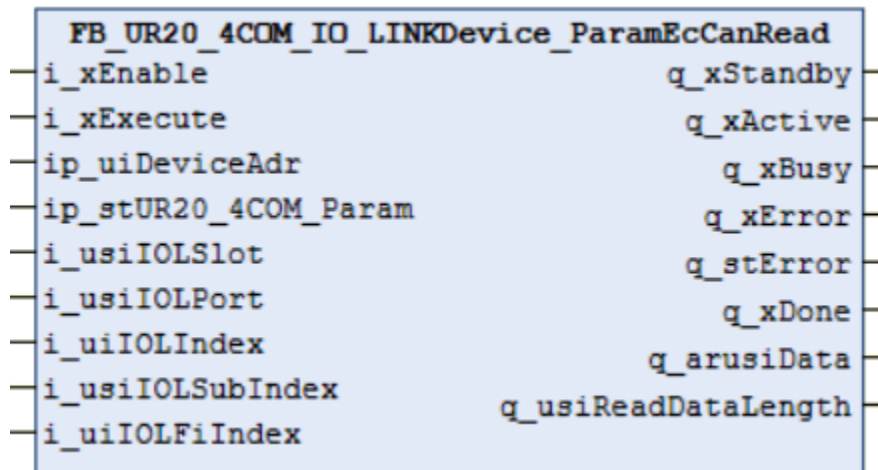
This library provides function blocks for the UR20-4COM-IO-LINK module connected via a CANopen-, EtherCAT-, or Modbus TCP fieldbus coupler. These function blocks enable your application to read or write data from / to an IO-Link slave device by non-cyclic communication over an IO-Link connection.

12.1 FB_UR20_4COM_IO_LINKDevice_ParamEcCanRead

Functional Description

The function block provides non-cyclic communication with an IO-Link slave device via an UR20-4COM-IO-LINK module attached to an EtherCAT fieldbus coupler. This FB reads data from an IO-Link slave device and transfers the data to your application.

The inputs of the function block must contain the position of the IO-Link device and which parameter (index + subindex) shall be read. Please read the IO-Link device manual for more information about the parameter index.



Possible Errors

Reason	Description
Invalid parameter	a. <code>ip_uiDeviceAdr</code> is not a valid EtherCat fieldbus coupler. b. <code>ip_stUR20_4COM_Param.udiReadTimeout</code> is zero. c. <code>ip_stUR20_4COM_Param.udiWriteTimeout</code> is zero. d. <code>ip_stUR20_4COM_Param.tTimeout</code> is zero.
Invalid process value	a. <code>i_usiIOLSlot</code> < 1 or > 64 b. <code>i_usiIOLPort</code> < 1 or > 4 c. <code>i_uiIOLIndex</code> < 1
read / write timeout	The CANopen-over-EtherCAT read / write operation timed out. Please check communication settings.
read / write error	The sub-FB reports an error during the IO-Link data write or read process in active state.
IO-Link call error	The UR20-4COM-IO-LINK module detected an error in the asynchronous communication to the IO-Link slave device.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_uiDeviceSlaveAdr	UINT	device slave address from Ethercat coupler
ip_stUR20_4COM_Param	ST_UR20_4COM_Param	this FB's parameters
i_usiIOLSlot	USINT	hardware slot of the UR20 module within your UR20 station; 1 ... 64
i_usiIOLPort	USINT	hardware port of the UR20 module where IO-Link device is connected; 1 ... 4
i_uiIOLIndex	UINT	IO-Link device parameter index (See IO-Link slave manual)
i_usiIOLSubIndex	USINT	IO-Link device parameter sub-index (See IO-Link slave manual)
i_uiIOLFiIndex	UINT	IO-Link FI Index (See IO-Link slave manual)

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arusiData	ARRAY [0..232] OF USINT	data from the IO-Link device (see IO-Link slave manual)
q_usiReadDataLength	USINT	number of bytes read from the IO-Link slave device

Structs

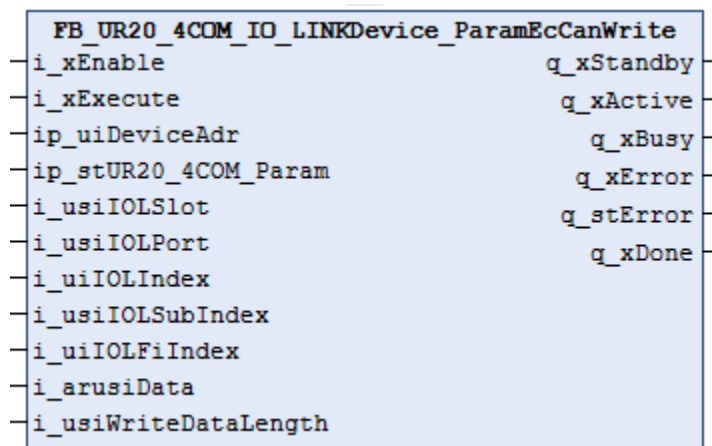
Name	Type	Comment
ST_UR20_4COM_Param	STRUCT	Module Parameters
tTimeout	TIME	The general timeout for EtherCAT / CANopen asynchronous SDO read / write operations. Default: TIME#50s0ms
udiReadTimeout	UDINT	The timeout in ms for the EtherCAT / CANopen asynchronous SDO read FB. Default: 50000
udiWriteTimeout	UDINT	The timeout in ms for the EtherCAT / CANopen asynchronous SDO write FB. Default: 50000

12.2 FB_UR20_4COM_IO_LINKDevice_ParamEcCanWrite

Functional Description

The function block provides non-cyclic communication with an IO-Link slave device via an UR20-4COM-IO-LINK module attached to an EtherCAT fieldbus coupler. This FB transfers write data from your application to an IO-Link slave device.

The inputs of the function block must contain the position of the IO-Link device and which parameter (index + subindex) shall be written. Please read the IO-Link device manual for more information about the parameter index.



Possible Errors

Reason	Description
invalid parameter	a. ip_uiDeviceAdr is not a valid EtherCat fieldbus coupler. b. ip_stUR20_4COM_Param.udiReadTimeout is zero. c. ip_stUR20_4COM_Param.udiWriteTimeout is zero. d. ip_stUR20_4COM_Param.tTimeout is zero.
invalid process value	a. i_usiIOLSlot < 1 or > 64 b. i_usiIOLPort < 1 or > 4 c. i_uiIOLIndex < 1 d. usiWriteDataLength = 0 or > 233
read / write timeout	The CANopen-over-EtherCAT read / write operation timed out. Please check communication settings.
read / write error	The sub-FB reports an error during the IO-Link data write or read process in active state.
IO-Link call error	The UR20-4COM-IO-LINK module detected an error in the asynchronous communication to the IO-Link slave device.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active

Name	Type	Comment
ip_uiDeviceAdr	UINT	Address of the CANopen or EtherCAT fieldbus coupler
ip_stUR20_4COM_Param	ST_UR20_4COM_Param	This FB's parameters
i_usiIOLSlot	USINT	hardware slot of the UR20 module within your UR20 station; 1 ... 64
i_usiIOLPort	USINT	hardware port of the UR20 module where IO-Link device is connected; 1 ... 4
i_uiIOLIndex	UINT	IO-Link device parameter index (See IO-Link slave manual)
i_usiIOLSubIndex	USINT	IO-Link device parameter sub-index (See IO-Link slave manual)
i_uiIOLFiIndex	UINT	IO-Link FI Index (See IO-Link slave manual)
i_arusiData	ARRAY [0..232] OF USINT	data to be written to the IO-Link device (see IO-Link slave manual)
i_usiWriteDataLength	USINT	data length to be written to the IO-Link device

Outputs

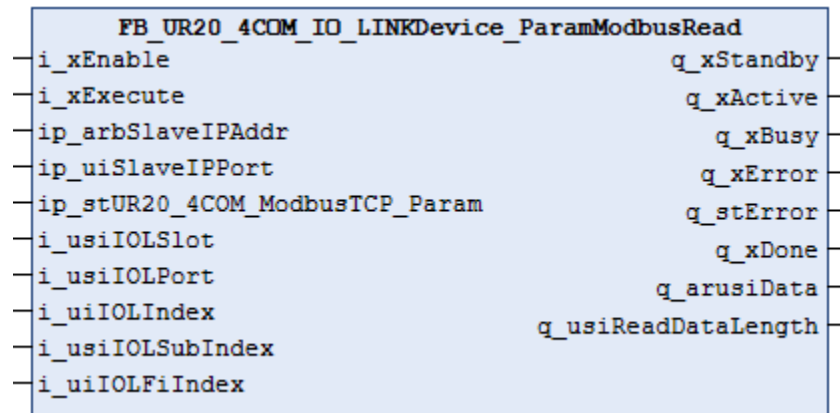
Name	Type	Comment
q_xStandby	BOOL	waiting for activation
q_xActive	BOOL	function block is activated
q_xBusy	BOOL	function block is activated and doing its supposed task
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

12.3 FB_UR20_4COM_IO_LINKDevice_ParamModbusRead

Functional Description

The function block provides non-cyclic communication with an UR20-4COM-IO-LINK module and an IO-Link slave device over Modbus TCP. The function block reads parameter data from the IO-Link slave device and transfers it to your application.

The inputs of the function block must contain the position of the IO-Link device and which parameter, addressed by IO-Link index and -subindex, shall be read. Please read the IO-Link device manual to get more information about the slave device's parameter indices and -subindices.



Possible Errors

Reason	Description
invalid parameter	a. ip_arbSlaveIPAdress[0] is zero. b. ip_uiPort is zero. c. ip_stUR20_4COM_ModbusTCP_Param.udiReadTimeout is zero. d. ip_stUR20_4COM_ModbusTCP_Param.udiWriteTimeout is zero. e. ip_stUR20_4COM_ModbusTCP_Param.tTimeout is zero. f. ip_stUR20_4COM_ModbusTCP_Param.tModbusTCPConnectTimeOut is zero.
invalid process value	a. i_usiIOLSlot is < 1 or > 64. b. i_usiIOLPort is < 1 or > 4. c. i_uiIOLIndex is < 1.
modbus TCP client init error	The Modbus TCP client FB has reported an error on entry to standby.
modbus TCP client operation error	The Modbus TCP client FB has reported an error during an ongoing IO-Link transfer.
read / write timeout	The read / write operation has timed out, please check communication settings.
read / write error	The sub-FB reports an error during the Modbus TCP data write or read process in active state.
IO-Link call returned an error	the IO-Link slave device reported an error.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_arbSlaveIPAddr	ARRAY [0..3] OF BYTE	IP address of the Modbus TCP fieldbus coupler, e.g. [192, 168, 1, 2]
ip_uiSlaveIPPort	UINT	the TCP port of the Modbus TCP fieldbus coupler
ip_stUR20_4COM_ModbusTCP_Param	ST_UR20_4COM_ModbusTCP_Param	this FB's parameters
i_usiIOLSlot	USINT	hardware slot of the station where the UR20 module is connected; 1 - 64
i_usiIOLPort	USINT	hardware port of the UR20 module where IO-Link device is connected; 1 - 4
i_uiIOLIndex	UINT	IO-Link device parameter index (See IO-Link slave manual)
i_usiIOLSubIndex	USINT	IO-Link device parameter sub-index (See IO-Link slave manual)
i_uiIOLFiIndex	UINT	IO-Link FI index (See IO-Link slave manual)

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arusiData	ARRAY [0..232] OF USINT	Data from the IO-Link device (see IO-Link slave manual)
q_usiReadDataLength	USINT	number of bytes read from the IO-Link slave device

Structs

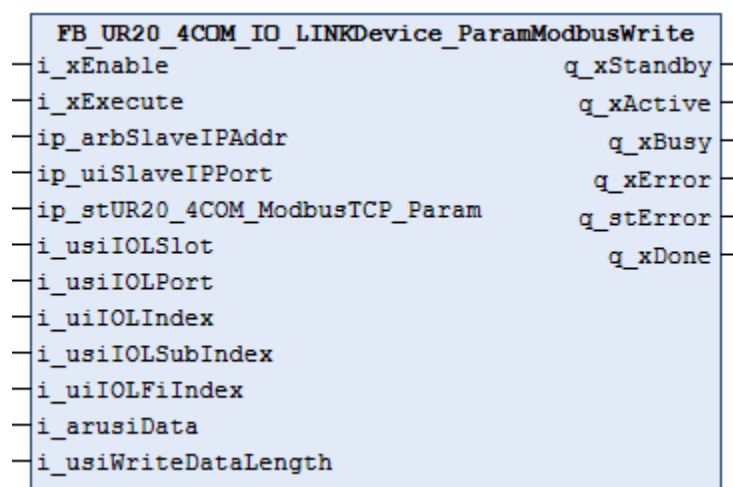
Name	Type	Comment
ST_UR20_4COM_ModbusTCP_Param	STRUCT	FB Parameter
tModbusTCPConnectTimeout	TIME	The timeout for connecting to the Modbus TCP server (aka slave) device. Default: TIME#1s0ms
tTimeout	TIME	The general timeout for EtherCAT / CANopen asynchronous SDO read / write operations. Default: TIME#50s0ms
udiReadTimeout	UDINT	The timeout in μ s for the Modbus TCP asynchronous read FB. Default: 50000000
udiWriteTimeout	UDINT	The timeout in μ s for the Modbus TCP asynchronous read FB. Default 50000000

12.4 FB_UR20_4COM_IO_LINKDevice_ParamModbusWrite

Functional Description

The function block provides non-cyclic communication with UR20-4COM-IO-LINK module and IO-Link slave device over Modbus TCP. The function block transfers parameter data from your application to a connected IO-Link slave device.

The inputs of the function block must contain the position of the IO-Link device and which parameter, addressed by IO-Link index and -subindex, shall be read. Please read the IO-Link device manual to get more information about the slave device's parameter indices and -subindices.



Possible Errors

Reason	Description
invalid parameter	a. ip_arbSlaveIPAddr[0] is zero. b. ip_uiPort is zero. c. ip_stUR20_4COM_ModbusTCP_Param.udiReadTimeout is zero. d. ip_stUR20_4COM_ModbusTCP_Param.udiWriteTimeout is zero. e. ip_stUR20_4COM_ModbusTCP_Param.tTimeout is zero. f. ip_stUR20_4COM_ModbusTCP_Param.tModbusTCPConnectTimeOut is zero.
invalid process value	a. i_usiIOLSlot is < 1 or > 64. b. i_usiIOLPort is < 1 or > 4. c. i_uiIOLIndex is < 1. d. usiWriteDataLength = 0 or > 233
modbus TCP client init error	The Modbus TCP client FB has reported an error on entry to standby.
modbus TCP client operation error	The Modbus TCP client FB has reported an error during an ongoing IO-Link transfer.
read / write timeout	The read / write operation has timed out, please check communication settings.
read / write error	The sub-FB reports an error during the Modbus TCP data write or read process in active state.
IO-Link call returned an error	the IO-Link slave device reported an error.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_arbSlaveIPAddr	ARRAY [0..3] OF BYTE	IP address of the ModBus TCP fieldbus coupler
ip_uiSlaveIPPort	UINT	the TCP port of the ModBus TCP fieldbus coupler
ip_stUR20_4COM_ModbusTCP_Param	ST_UR20_4COM_ModBusTCP_Param	this FB's parameters
i_usiIOLSlot	USINT	hardware slot of the station where the UR20 module is connected; 1 ... 64
i_usiIOLPort	USINT	hardware port of the UR20 module where IO-Link device is connected; 1 ... 4
i_uiIOLIndex	UINT	IO-Link device parameter index (See IO-Link slave manual)
i_usiIOLSubIndex	USINT	IO-Link device parameter sub-index (See IO-Link slave manual)
i_uiIOLFiIndex	UINT	IO-Link FI index (See IO-Link slave manual)
i_arusiData	ARRAY [0..232] OF USINT	Data to be written to the IO-Link device (see IO-Link slave manual)
i_usiWriteDataLength	USINT	Data length to be written to the IO-Link device

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

Structs

Name	Type	Comment
ST_UR20_4COM_ModBusTCP_Param	STRUCT	FB Parameter
tModbusTCPConnectTimeOut	TIME	The timeout for connecting to the ModBus TCP server (aka slave) device. Default: TIME#1s0ms
tTimeout	TIME	The general timeout for EtherCAT / CANopen asynchronous SDO read / write operations. Default: TIME#50s0ms
udiReadTimeout	UDINT	The timeout in μ s for the Modbus TCP asynchronous read FB. Default: 50000000
udiWriteTimeout	UDINT	The timeout in μ s for the Modbus TCP asynchronous read FB. Default 50000000

13 libWiUr20AiSg

This library provides parametrization and operation function blocks for the UR20_2AI_SG_24_DIAG two channel strain gauge sensor module. Please also refer to the UR20_2AI_SG_24_DIAG's description in the UR20 handbook available on the Weidmüller website.

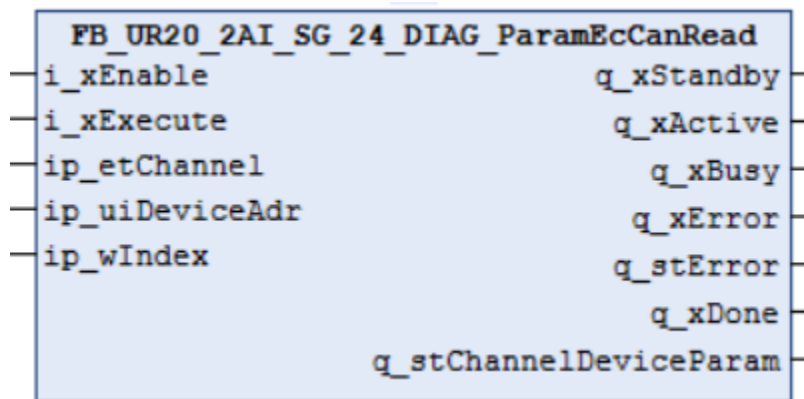
13.1 FB_UR20_2AI_SG_24_DIAG_ParamEcCanRead

Functional Description

This function block reads one channel's device parameters from an UR20_2AI_SG_24_DIAG module attached to a PLC via an UR20 EtherCAT or CANopen fieldbus coupler. Select a channel on `ip_etChannel`, enable and activate the function block (FB). You find the UR20_2AI_SG_24_DIAG module's actual device parameters in `q_stChannelDeviceParam`, after the FB has set `q_xDone` to **true** to signal completion of its task.

A note regarding the input `ip_uiDeviceAdr`: To find the required value in your project, open the EtherCAT coupler device in the devices tree. Then open the tab 'General' and use the value shown as 'EtherCAT address'.

A note regarding the input `ip_wIndex`: To find the required value in your project, open the desired UR20_2AI_SG_24_DIAG module in your devices tree. In the tab 'Startup Parameters', find the column labelled 'Index:Subindex'. Use the value shown for 'Index'.



Possible Errors

Reason	Description
EtherCAT/CAN parameter read error	EtherCAT/CAN parametrization function block has error, see CoDeSys help of ETC_CO_SdoReadDWord for more details.
<code>ip_uiDeviceAdr</code> not connected	A variable must be connected to this mandatory input parameter.
<code>ip_wIndex</code> not connected	A variable must be connected to this mandatory input parameter.
invalid parameter	a. <code>ip_uiDeviceAdr</code> is zero. Use a non-zero value. b. <code>ip_wIndex</code> is zero. Use a non-zero value.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_etChannel	ET_UR20_2AI_SG_24_DIAG_Channel	channel to be parametrized channel zero / channel one
ip_uiDeviceAdr	UINT	Address of CanOpen or EtherCAT device [coupler]
ip_wIndex	WORD	start address of the module

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_stChannelDeviceParam	ST_UR20_2AI_SG_24_DIAG_DeviceParam	Device parameters for one strain gauge module channel

13.2 FB_UR20_2AI_SG_24_DIAG_ParamEcCanWrite

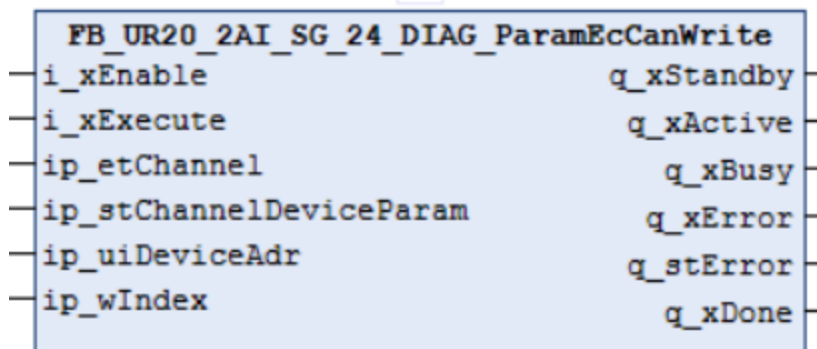
Functional Description

This function block writes one channel's device parameters to an UR20_2AI_SG_24_DIAG module attached to a PLC via an UR20 EtherCAT or CANopen fieldbus coupler.

Select a channel on ip_etChannel and enable the FB by setting i_xEnable to **true**. The FB will check the device parameter values provided in ip_stChannelDeviceParam and indicate invalid values as an error or indicate a successful device parameter check by setting q_xStandby to **true**. Now activate the function block by setting i_xExecute to **true**. The function block will write the device parameters for the selected channel to the UR20_2AI_SG_24_DIAG module. After the function block has finished the transfer, it sets q_xDone to **true** to signal completion of its task.

A note regarding the input ip_uiDeviceAdr: To find the required value in your project, open the EtherCAT coupler device in the devices tree. Then open the tab 'General' and use the value shown as 'EtherCAT address'.

A note regarding the input ip_wIndex: To find the required value in your project, open the desired UR20_2AI_SG_24_DIAG module in your devices tree. In the tab 'Startup Parameters', find the column labelled 'Index:Subindex'. Use the value shown for 'Index'.



Possible Errors

Reason	Description
EtherCAT/CAN parameter write error	EtherCAT/CAN parametrization function block has error, see CoDeSys help of ETC_CO_SdoWriteDWord for more details.
ip_uiDeviceAdr not connected	A variable must be connected to this mandatory input parameter.
ip_wIndex not connected	A variable must be connected to this mandatory input parameter.
invalid parameter	a) ip_uiDeviceAdr is zero. Use a non-zero value. b) ip_wIndex is zero. Use a non-zero value.
invalid process value	a) ip_stChannelDeviceParam.diSensorSensitivity < 500000 b) ip_stChannelDeviceParam.diSensorSensitivity > 960000000 Use a value > 500000 and < 960000000.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_etChannel	ET_UR20_2AI_SG_24_DIAG_Channel	channel to be parametrized channel zero / channel one
ip_stChannelDeviceParam	ST_UR20_2AI_SG_24_DIAG_DeviceParam	device parameters for one strain gauge module channel
ip_uiDeviceAdr	UINT	Address of CanOpen or EtherCAT device [coupler]
ip_wIndex	WORD	start address of the module

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_xError	BOOL	function block is in error state

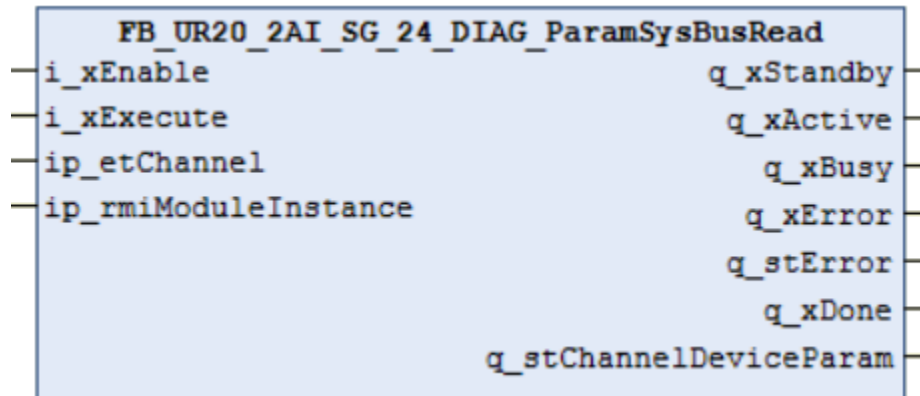
Name	Type	Comment
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

13.3 FB_UR20_2AI_SG_24_DIAG_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block is intended for use with the UR20-2AI-SG-24-DIAG U-Remote module.

Please observe that this module has two channels; You need to select the channel to read from in ip_etChannel before you activate the function block. For the same reason, the name of the parameter output deviates from the common description and is q_stChannelDeviceParam.



Specific Inputs

Name	Type	Comment
ip_etChannel	ET_UR20_2AI_SG_24_DIAG_Channel	channel to be read from: channel zero / channel one

Specific Outputs

Name	Type	Comment
q_stChannelDeviceParam	ST_UR20_2AI_SG_24_DIAG_DeviceParam	device parameters for one strain gauge module channel

Structs

Name	Type	Comment
ST_UR20_2AI_SG_24_DIAG_DeviceParam	STRUCT	This struct holds the device parameters for one channel of the UR20_2AI_SG_24_DIAG module.
etFrequencySuppression	ET_UR20_AI_UI_FrequencySuppression	selects the signal frequency suppression

etConnectionType	ET_UR20_2AI_SG_24_DIAG_ConnectionType	connection type: 4-Channel / 6-Channel
etConversionTime	ET_UR20_2AI_SG_24_DIAG_ConversionTime	conversion time according to the u-remote manual
etChannelDiagnostic	ET_UR20_2AI_SG_24_DIAG_ChannelDiagnostic	activate channel diagnostic / deactivate channel diagnostic
etTareFunction	ET_UR20_2AI_SG_24_DIAG_TareFunction	set up tare function: deactivate / digital input / software / digital input AND software / digital input OR software
diSensorSensitivity	DINT	sensitivity of the connected sensor: 500,000 ... 960,000,000
diFullScale	DINT	Full scale weight, defines upper measurement range limit
diOffset	DINT	offset value to adjust the zero position

13.4 FB_UR20_2AI_SG_24_DIAG_ParamSysBusWrite

The function block derives from the FB_LevelControlledFiniteBehavior model. It writes the parameter set of _one channel_ in ip_stChannelDeviceParam to a UR20-2AI-SG-24-DIAG u-remote module connected to the System Bus of a u-control with u-OS.

The user enables the FB. The FB checks that ip_rmiModuleInstance is a valid module instance and that it references a module of the correct type. Then the FB enters the state **standby**. In state **standby**, the FB checks readiness of its communication sub-FB.

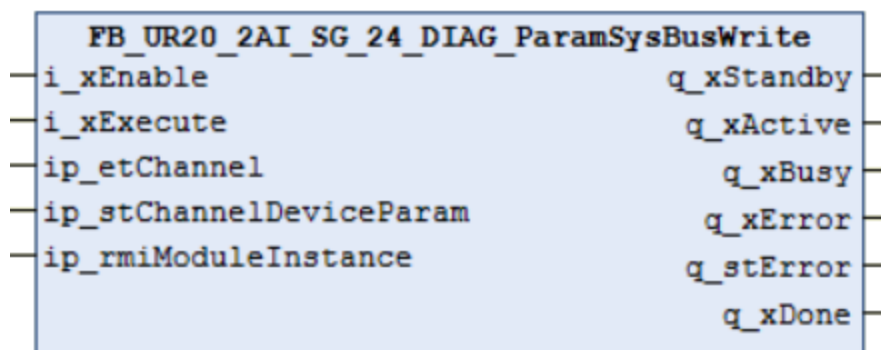
The user selects a channel via ip_etChannel and activates the FB. The FB shadow-copies ip_etChannel. It checks validity of the parameter set in ip_stChannelDeviceParam, then it shadow-copies internally the parameter set in ip_stChannelDeviceParam and enters the state **active**. In the state **active**, the FB reads the parameter set from the module. Then it translates the shadow-copied parameter set and merges it into the byte array it has read from the module and sends that to the SG module via a communication sub-FB. If any checks fail or the sub-fbs report an error, then the FB enters its state **error**. The FB leaves the state **error** after the user deactivates and disables the FB.

To send another parameter set, the user deactivates the FB. The FB returns to its state **standby**. The user applies a new parameter set to the input ip_stChannelDeviceParam, then the user activates the FB, again. The FB then translates and sends the new parameter set to the module.

Note: Assign the instance of your UR20-2AI-SG-24-DIAG in the device tree to ip_rmiModuleInstance. Here is an example:

```
fbParamWrite(ip_rmiModuleInstance := UR20_2AI_SG_24_DIAG_1);
```

The function block is intended for use with the UR20-2AI-SG-24-DIAG U-Remote module.



Possible Errors

Reason	Description
invalid module reference	invalid input parameter "ip_rmiModuleInstance"
wrong module TypeName	ip_rmiModuleInstance references the wrong module type
reset error	The sub-FB exposed an error during deactivation.
invalid process value	a. ip_stChannelDeviceParam.diSensorSensitivity < 500000 b. ip_stChannelDeviceParam.diSensorSensitivity > 960000000 Use a value >= 500000 and =< 960000000.
sub-fb error	fbAsyncParamRead reports an error. Refer to IoDrvUR20Control -> Enums -> Error.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_rmiModuleInstance	REFERENCE TO IoDrvUR20Control.IoDrvUR20ModuleDiag	module instance to read the parameters from
ip_etChannel	ET_UR20_2AI_SG_24_DIAG_Channel	channel to be parametrized channel zero / channel one
ip_stChannelDeviceParam	ST_UR20_2AI_SG_24_DIAG_DeviceParam	device parameters for one strain gauge module channel

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information

Name	Type	Comment
q_xDone	BOOL	execution has come to its supposed end

13.5 FB_UR20_2AI_SG_24_DIAG_Calib

Functional Description

This function block provides a calibration procedure for one channel of the UR20_2AI_SG_24_DIAG module. After the user has enabled the FB by setting `i_xEnable` to **true**, the function block checks the device parameters in `ip_stInitialChannelDeviceParam`, the FB's parameters in `ip_stCalibParam`, and checks that `i_fbParamWrite` is a valid reference. If any reference or parameter test fails, the FB indicates this in the state **Error** and sets `q_xError` and additional error information in `q_stError`. If all parameter and reference tests are ok, the FB transitions to the state **Standby**. On entry into the state **Standby**, the FB enables the `i_fbParamWrite` sub-FB and verifies that the `i_fbParamWrite` confirms a successful entry into its state **Standby**. Then the FB indicates its state **Standby** by setting `q_xStandby` to **true**.

After the user has activated the FB `FB_UR20_2AI_SG_24_DIAG_Calib` by setting `i_xExecute` to **true**, the FB will begin the calibration procedure for the channel selected in `ip_stCalibParam.etChannel`. To do so, the FB guides the user through the calibration procedure with short texts in `q_strUserTask` and waits for the user to confirm completion of these tasks with a rising edge on `i_xNextStep`. Principally, the calibration procedure first determines an interim offset value that brings the output of a channel connected to an unloaded strain gauge to zero. Then the procedure evaluates the measurement value indicated by the UR20_2AI_SG_24_DIAG module while a known test weight loads the strain gauge. The FB checks that the measurement value reported by the UR20_2AI_SG_24_DIAG module does not indicate a saturation of that channel, then it calculates corrected sensitivity and offset parameter values and writes these to the UR20_2AI_SG_24_DIAG module.

The FB waits for the measurement value to reach a steady state after a load change. That is, it takes a measurement value, waits for `ip_stInitialChannelDeviceParam.etConversionTime` then takes a measurement value again. If the absolute difference is less than `ip_stCalibParam.diSteadyDeviationThreshold`, the FB accepts this as steady state. If the absolute difference is above `ip_stCalibParam.diSteadyDeviationThreshold`, it repeats the above for up to `ip_stCalibParam.bNumberOfSteadyCycles` counts. If the measurement value is not steady after that many tries, the FB reports an error.

Note that the conversion time acts as a low pass filter time on the measurement value. A low conversion time will result in faster response to load changes but can lead to failure of the calibration process because the measurement signal keeps changing, e.g due to shaking or vibration of the load or load cell.



The above description of the calibration procedure is valid for UR20-2AI-SG-DIAG firmware from version 1.00.06 on. If your module has a firmware version before 1.00.06, please update the module to at least version 1.00.06.

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		is 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

The FB indicates completion of the calibration procedure by setting `q_xDone` to **true**. The outputs `q_diCorrectedSensorSensitivity` and `q_diCorrectedOffset` keep showing the corrected values until the user deactivates and/or disables the FB.

The FB uses the following formula to calculate the interim offset to reach zero displayed value:

```
interim offset = measurement value + initial offset
```

The FB uses the following formula to calculate the corrected offset and sensitivity:

```

correction factor    =    displayed value
                        -----
                        calibration normal

corrected sensitivity = initial sensitivity * correction factor

corrected offset     =    interim offset
                        -----
                        correction factor
  
```

Note that FB_UR20_2AI_SG_24_DIAG_Calib needs to write device parameters to the UR20_2AI_SG_24_DIAG module. Since knowledge of the actual field bus in the user's system is not necessary for FB_UR20_2AI_SG_24_DIAG_Calib's function, we abstract the FB_UR20_2AI_SG_24_DIAG_Calib from the fieldbus used:

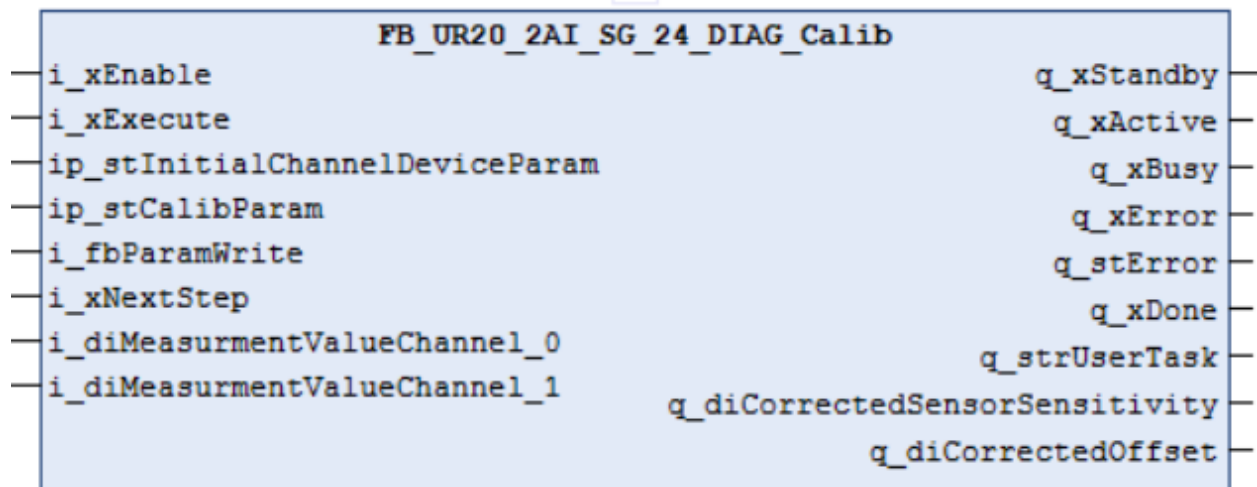
The user instantiates a parameter write FB suitable for his fieldbus and then assigns it to FB_UR20_2AI_SG_24_DIAG_Calib's i_fbParamWrite input. Here is an example, assuming the user's system has an EtherCAT field bus:

```
VAR
    fbParamWrite : FB_UR20_2AI_SG_24_DIAG_ParamEcCanWrite;
END_VAR

fbCalib(i_fbParamWrite := fbParamWrite,
    <the other input and output assignments follow here..> );
```

Also do not forget to assign any parameters and input values your parameter write FB needs. In our example, the fbParamWrite : FB_UR20_2AI_SG_24_DIAG_ParamEcCanWrite needs a device address and module index :

```
IF (xInitDone = FALSE) THEN
    xInitDone := TRUE;
    fbParamWrite(ip_uiDeviceAdr := UR20_FBC_EC.PhysSlaveAddr, ip_wIndex :=
16#8000);
END_IF;
```



Possible Errors

Reason	Description
calculation of corrected sensitivity or offset out of range	A new sensitivity or offset is calculated, that is out of range to be set as a parameter for the parametrization of the module.

Reason	Description
i_fbParamWrite parameter error	i_fbParamWrite is not a valid reference to an instance of FB_UR20_2AI_SG_24_DIAG_ParamAnyWrite.
device parameter error	ip_stInitialChannelDeviceParam.diSensorSensitivity < 500000 or ip_stInitialChannelDeviceParam.diSensorSensitivity > 960000000
fb parameter error	<ul style="list-style-type: none"> a. ip_st_CalibParam.diReferenceWeight > ip_stInitialChannelDeviceParam.diFullScale b. ip_st_CalibParam.diReferenceWeight = 0 c. ip_st_CalibParam.diReferenceWeight has different numerical sign than ip_stInitialChannelDeviceParam.diFullScale d. ip_st_CalibParam.bNumberOfSteadyCycles = 0 e. ip_stCalibParam.diSteadyDeviationThreshold < 0
i_fbParamWrite instance error	The FB_UR20_2AI_SG_24_DIAG_Calib has forwarded an error reported by the FB param write instance connected to i_fbParamWrite.
calculated sensitivity or offset out of range	<ul style="list-style-type: none"> a. rCorrectedSensitivity < 500000 b. rCorrectedSensitivity > 960000000 c. rCorrectedOffset < -2147483648 d. rCorrectedOffset > 2147483647 <p>Check for bad initial parameters, a broken and/or misconnected load cell or mismatching actual vs. parametrized reference weight.</p>
calculated correction factor is too small	rCorrectionFactor = 0. Check for a correctly placed reference weight, broken connections, or a defective load cell.
timeout: measurement signal not steady	The measurement signal kept changing for more than ip_st_CalibParam.bNumberOfSteadyCycles times the parametrized conversion time. Try a higher conversion time or increase ip_stInitialChannelDeviceParam.etConversionTime. Check for a shaking or vibrating load and / or load cell or electrical noise.
measurement value indicates channel saturation	the load cell signal has exceeded +/- 150 mV. Check for overload, wrong connection, or a defective load cell.
unknown state in steady measurement state machine	The FB has an internal error. Please contact the Weidmueller service team and provide the content of q_stError.
unknown state in calibration state machine	The FB has an internal error. Please contact the Weidmueller service team and provide the content of q_stError.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_stInitialChannelDeviceParam	ST_UR20_2AI_SG_24_DIAG_DeviceParam	device parameters for one strain gauge module channel
ip_stCalibParam	ST_UR20_2AI_SG_24_DIAG_CalibParams	parameters for operation of this function block

Name	Type	Comment
i_fbParamWrite	REFERENCE TO FB_UR20_2AI_SG_24_DIAG_ ParamAnyWrite	reference to a device parameter write function block
i_xNextStep	BOOL	a rising edge on this bool toggles the next calibration step
i_diMeasurmentValue Channel_0	DINT	process data of channel 0 from module
i_diMeasurmentValue Channel_1	DINT	process data of channel 1 from module

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_strUserTask	STRING	Short textual description of required user interaction for next calibration step (remove or place reference weight)
q_diCorrectedSensor Sensitivity	DINT	corrected sensitivity value, valid after successful callibration
q_diCorrectedOffset	DINT	corrected sensitivity value, valid after successful callibration

Structs

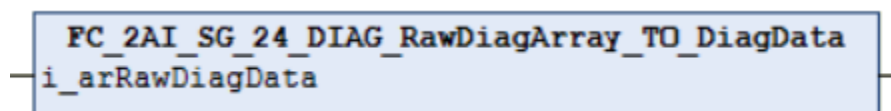
Name	Type	Comment
ST_UR20_2AI_SG_24_DIAG_Device Param	STRUCT	2AI-SG module parameter
etConnectionType	ET_UR20_2AI_SG_24_DIAG_ ConnectionType	connection type: 4-Channel / 6-Channel
etConversionTime	ET_UR20_2AI_SG_24_DIAG_ ConversionTime	conversion time according to the u-remote manual
etChannelDiagnostic	ET_UR20_2AI_SG_24_DIAG_ ChannelDiagnostic	activate channel diagnostic / deactivate channel diagnostic
etTareFunction	ET_UR20_2AI_SG_24_DIAG_ TareFunction	set up tare function: deactivate / digital input / software / digital input AND software / digital input OR software
diSensorSensitivity	DINT	sensitivity of the connected sensor: 500,000 ... 960,000,000
diFullScale	DINT	Full scale weight, defines upper measurement range limit
diOffset	DINT	offset value to adjust the zero position
ST_UR20_2AI_SG_24_DIAG_CalibP arams	STRUCT	Parameters for calibration of 2AI-SG module

diReferenceWeight	DINT	reference weight for calibration. Must be smaller than ip_stInitialChannelDeviceParam.diFullScale
bNumberOfSteadyCycles	BYTE	FB_UR20_2AI_SG_24_DIAG_Calib waits this many times the conversion time for the UR20_2AI_SG_24_DIAG's measurement value to become steady after a weight change
diSteadyDeviationThreshold	DINT	maximum deviation that FB_UR20_2AI_SG_24_DIAG_Calib accepts as UR20_2AI_SG_24_DIAG's measurement value steady state

13.6 FC_2AI_SG_24_DIAG_RawDiagArray_TO_DiagData

Please consult the Application Note AN0107v1-UC20-u-OS CODESYS Diagnosis- and Process Alarms for an overview over the alarm message transfer.

This function converts a raw diagnosis alarm message array to a more readable format.



Inputs

Name	Type	Comment
i_arRawDiagData	ARRAY [0..46] OF BYTE	raw diagnostic alarm message from UR20 module

Return

Name	Type	Comment
FC_2AI_SG_24_DIAG_RawDiagArray_TO_DiagData	ST_UR20_2AI_SG_24_DIAG_DiagData	The diagnostic alarm message in a struct.

Structs

Name	Type	Comment
ST_UR20_2AI_SG_24_DIAG_DiagData	STRUCT	This struct represents the module-specific part of a diagnostic message of a UR20-2AI-SG-24-DIAG module. Please find the common part in libWiUr20Diag -> ST_UR20_DiagDataUremoteClass.
stErrorIndicator	ST_UR20_DiagDataErrorIndicator	Byte 0: error indicator
enModuleType	ET_UR20_ModuleType	Byte 1: default value 0x0F
bErrorByte2	BYTE	Byte 2: default value 0
stErrorByte3	ST_UR20_DiagDataErrorType	Byte 3: default value 0
enChannelType	ET_UR20_ChannelType	Byte 4: default value 0x70
bNumberOfDiagBits	BYTE	Byte 5: number of diagnostic bit per channel
bNumberOfChannels	BYTE	Byte 6: number of similar channels per module

stErrorChannel	ST_UR20_DiagDataErrorAtChannel	Byte 7: indicates which channel has the error
arbChannelError	ARRAY [0..2] OF BYTE	BYTE 8..10: expanded diagnosis
dwTimeStamp	DWORD	BYTE 43..46: time stamp μ s
arst2AISGDiagErrorChannel	ARRAY [0..1] OF ST_UR20_2AI_SG_24_DIAG_DiagChannelError	Byte 11,12, channel 0,1 error diagnostic flags

Structs

Name	Type	Comment
ST_UR20_2AI_SG_24_DIAG_DiagChannelError	STRUCT	This struct represents the channel error bits of a UR20-2AI-SG-24-DIAG module.
xParameterError	BOOL	the module has received invalid parameters
xOverload	BOOL	the channel reports an overload
xReserved	BOOL	reserved
xShortCircuit	BOOL	the module has detected a short circuit
xLineBreak	BOOL	the module has detected a line break
xReserved1	BOOL	reserved
xReserved2	BOOL	reserved
xReserved3	BOOL	reserved

14 libWiUr20ComCanOpen

This library provides function blocks for the UR20-1COM-CANOPEN module connected via a u-remote fieldbus coupler. These function blocks enable your application to read or write data from / to a CAN/CANopen slave device.

Note: To let the module receive any CAN messages it is necessary to activate and configure the RX Filter, RX Can ID and RX Mask parameter.

	RX Filter 1 ... 8	enabled (0) / disabled (1)	disabled
0	RX CAN ID 1 ... 8	0 ... 2047	0
	RX Maske 1 ... 8	0 ... 2047	2047

e.g. to read all CAN messages on the fieldbus set RX Filter[1] to enabled, RX CAN ID[1] to 0 and RX Maske[1] to 0. Rx CAN id and Rx Mask are logically “**AND**” linked.

Available hardware description files for the UR20-1COM-CANOPEN and the parametrization of the module lets the user configure the process input- and process output data length of the module to a choice of fixed values.

14.1 FB_UR20_1Com_CANopen_Transparent

Functional Description

The function block provides an interface to the UR20-1COM-CANopen module for the transparent mode. The FB establishes a CAN communication to one or more CAN devices connected via the UR20-1COM-CANopen module.

The plc application on the one hand and the function block on the other hand exchange CAN messages via two arrays:

- The fixed-size array `iq_arstCanTelegramTx` holds the CAN messages to be sent. The maximum size is 12 messages in one cycle.
- The variable-size array `iq_arstCanTelegramRx` holds the received CAN messages. The size of `iq_arstCanTelegramRx` must be greater than or equal to the maximum amount of CAN msg per cycle according to the chosen process image data size.

The function block expects and provides the hardware process data in two arrays of variable size: `iq_arbHwProcessInputData` and `iq_arbHwProcessOutputData`.

The process data width is variable and depends on the device description file and parametrization of the module. Weidmueller offers several device description files with different process data sizes. The decision which data width to use depends on the customer application. One 11 bit-ID CAN message takes 10 bytes of process data. Accordingly, it is possible to read/send 12 messages per cycle if the user selects the maximum of 128 bytes for the process data size.

After the user has enabled the FB by setting `i_xEnable` to **true**, the function block checks the sizes of the hardware process data arrays `iq_arbHwProcessInputData`, `iq_arbHwProcessOutputData` and of the received CAN messages array `iq_arstCanTelegramRx`.

If any size check fails, the FB indicates this in the state **Error** and sets `q_xError` and additional error information in `q_stError`. If all size checks are ok, the FB transitions to the state **Standby**. On entry into the state **Standby**, the FB issues a Reset Module command to the UR20-1COM-CANopen module. After the module has confirmed completion of the Reset Module command,

the FB indicates its state **Standby** by setting `q_xStandby` to **true**. If any error occurs during the module reset or if the module reset times out, the FB enters its state **Error**, instead and provides detailed error information on its interface.

After the user has activated the `FB_UR20_1Com_CANOpen_Transparent` by setting `i_xExecute` to **true**, the FB will enter its state **Active**. Use the variables `xTransmitMessage` and `xTransmitDone` in the struct `ST_UR20_1Com_CANOpen_CanTelegram` to control the data transfer:

- pass the CAN messages to be sent in the array `iq_arstCanTelegramTx`
- set `xTransmitMessage` in `ST_UR20_1Com_CAN_CanTelegram` to send a message.
- The FB confirms sending of the message by setting `xTransmitDone` to true.

Receive telegrams:

To receive CAN messages with the `UR20-1COM-CANOpen` module it is necessary to activate and configure the RX Filter, RX Can ID and RX Mask parameter. e.g. to read all CAN messages on the fieldbus set `RX Filter[1]` to enabled, `RX CAN ID[1]` to 0 and `RX Maske[1]` to 0. Rx CAN id and Rx Mask are logically "AND" linked.

The FB transfers CAN messages the `UR20-1COM-CANOpen` module has received into the array `iq_arstCanTelegramRx`. The user sets the variable `i_xResetTelegramRxCnt` to reset the `q_diTelegramCounter`. This acknowledges that the user has processed the received data.

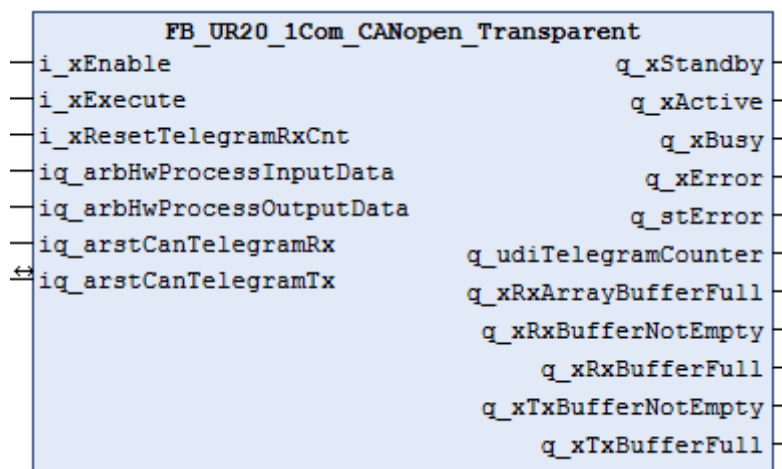
- the array `iq_arstCanTelegramRx` contains the received telegrams.
- The output `q_diTelegramCounter` shows the current number of received telegrams, it is also the current iterator of the input data array.
- the input `i_xResetTelegramRxCnt` resets the telegram counter `q_diTelegramCounter`.
- if the receive array `iq_arstCanTelegramRx` is full, the FB indicates this by setting the output `q_diRxArrayBufferFull` to true. The FB cannot receive any more messages until the user sets `i_xResetTelegramRxCnt` to true.

The Fb output `q_xRxBufferNotEmpty` indicates whether new data is available in the receive buffer of the module.

The Fb indicates that the receive buffer is full by setting the output `q_xRxBufferFull` to true. In this situation, the `UR20-1COM-CANOpen` cannot store more new CAN messages until the user has fetched messages from the module and cleared the receive buffer.

To reset a function block error, it is necessary to reset the inputs `i_xExecute` and `i_xEnable`.

Please refer to the u-remote manual for further details.



Possible Errors

Reason	Description
Invalid Parameter	Check the input parameter of the function block. The FB's parameters must be: <ul style="list-style-type: none"> a. size of <code>iq_arbHwProcessInputData</code> = <code>iq_arbHwProcessOutputData</code> b. array size of <code>iq_arbHwProcessInputData</code> = [128],[64],[34] or [18] c. array size of <code>iq_arstCanTelegramRx</code> >= max amount of possible can msg per cycle
input data conversion	The input data is corrupt and cannot be processed further. Check the module's process data mapping. Check the detailed error information.
output data conversion	The output data is corrupt. Check the configured CAN messages and check the detailed error information.
reading and sending procedure timed out	No response from the module within timeframe for the writing and sending sequence.
shutdown procedure timed out	No response from the module within timeframe for the shutdown sequence.
Module reset timed out	No response from the module while the RX and TX buffers are flushed.

Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active
<code>i_xResetTelegramRxCnt</code>	BOOL	acknowledge user has processed the received data

Outputs

Name	Type	Comment
<code>q_xStandby</code>	BOOL	waiting for activation

Name	Type	Comment
q_xActive	BOOL	function block is activated
q_xBusy	BOOL	function block is activated and doing its supposed task
q_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_udiTelegramCounter	UDINT	information about received telegrams
q_xRxArrayBufferFull	BOOL	information for user receive array buffer full
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

InOut

Name	Type	Comment
iq_arbHwProcessInputData	ARRAY [*] OF BYTE	Array of byte for process data input from hardware module
iq_arbHwProcessOutputData	ARRAY [*] OF BYTE	Array of byte for process data output to the hardware module
iq_arstCanTelegramRx	ARRAY [*] OF ST_UR20_1Com_CANopen_CanTelegram	CAN telegrams received by the module
iq_arstCanTelegramTx	ARRAY [0..11] OF ST_UR20_1Com_CANopen_CanTelegram	CAN telegrams to be sent

Structs

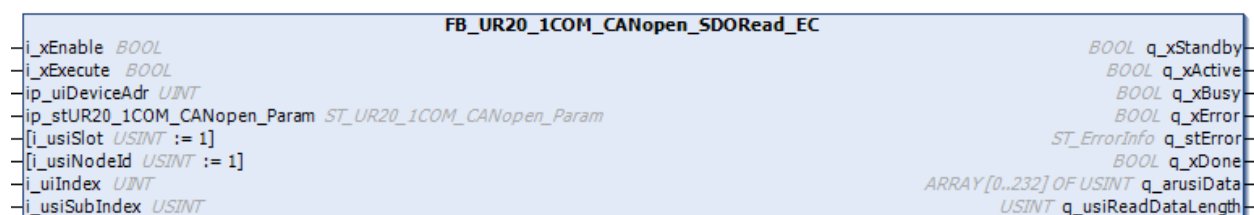
Name	Type	Comment
ST_UR20_1Com_CANopen_CanTelegram	STRUCT	CAN telegram data
uiCanIdentifier	UINT	CAN identifier for output data 0..2047
xCanRTR	BOOL	RTR Bit in output data (Remote Transmission Req.)
usiDataLength	USINT	amount of user data to be sent in current message. value 0..8 bytes
arbCanData	ARRAY [0..7] OF BYTE	array of user data, 8 bytes
xTransmitMessage	BOOL	if send flag is true the message will be transmitted
xTransmitDone	BOOL	true when the message has been sent

14.2 FB_UR20_1Com_CANopen_SDORRead_EC

Functional Description

The function block provides non-cyclic communication with a CANopen slave device via an UR20-1COM-CANopen module attached to an EtherCAT fieldbus coupler. This FB reads data from a CANopen slave device and transfers the data to your application.

The inputs of the function block must contain the EtherCAT address of the coupler, the position of the CANopen module (slot), the node id and which parameter (index + subindex) shall be read from the slave device.



Possible Errors

Reason	Description
invalid parameter	a. ip_uiDeviceAdr is not a valid EtherCat fieldbus address. b. ip_stUR20_1COM_CANopen_Param.udiReadTimeout is zero. c. ip_stUR20_1COM_CANopen_Param.udiWriteTimeout is zero. d. ip_stUR20_1COM_CANopen_Param.tTimeout is zero.
invalid process value	a. i_usiSlot < 1 or > 64 b. i_usiNodeId < 1 or > 127 c. i_uiIndex < 1
request timeout	The SDO read / write operation timed out. Please check communication settings.
read / write error	The sub-fb reports an error during the data write or read process in active state.
call error at request	The UR20-1COM-CANopen module detected an error in the asynchronous communication to the CANopen slave device.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_uiDeviceAdr	UINT	device slave address from Ethercat coupler
ip_stUR20_1COM_CANopen_Param	ST_UR20_1COM_CANopen_Param	this FB's parameters

Name	Type	Comment
i_usiSlot	USINT	hardware slot of the UR20 module within your UR20 station; 1 ...64
i_usiNodeId	USINT	Node ID of the CANopen slave device
i_uiIndex	UINT	device parameter index
i_usiSubIndex	USINT	device parameter sub-index

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arusiData	ARRAY [0..232] OF USINT	data from the CANopen device
q_usiReadDataLength	USINT	number of bytes read from the CANopen slave device

Structs

Name	Type	Comment
ST_UR20_1COM_CANopen_Param	STRUCT	Module Parameters
tTimeout	TIME	The general timeout for asynchronous SDO read / write operations. Default: TIME#50s0ms
udiReadTimeout	UDINT	The timeout in ms for the asynchronous SDO read FB. Default: 50000
udiWriteTimeout	UDINT	The timeout in ms for the asynchronous SDO write FB. Default: 50000

14.3 FB_UR20_1Com_CANopen_SDOWrite_EC

Functional Description

The function block provides non-cyclic communication with a CANopen slave device via an UR20-1COM-CANopen module attached to an EtherCAT fieldbus coupler. This FB transfers write data from your application to a CANopen slave device.

The inputs of the function block must contain the EtherCAT address of the coupler, the position of the CANopen module (slot), the node id and which parameter (index + subindex) shall be read from the slave device.

FB_UR20_1COM_CANopen_SDOWrite_EC		
i_xEnable	BOOL	BOOL q_xStandby
i_xExecute	BOOL	BOOL q_xActive
ip_uiDeviceAdr	UINT	BOOL q_xBusy
ip_stUR20_1COM_CANopen_Param	ST_UR20_1COM_CANopen_Param	BOOL q_xError
[i_usiSlot	USINT := 1]	ST_ErrorInfo q_stError
[i_usiNodeId	USINT := 1]	BOOL q_xDone
i_uiIndex	UINT	
i_usiSubIndex	USINT	
i_arusiData	ARRAY[0..232] OF USINT	
i_usiWriteDataLength	USINT	

Possible Errors

Reason	Description
invalid parameter	a. ip_uiDeviceAdr is not a valid EtherCat fieldbus address. b. ip_stUR20_1COM_CANopen_Param.udiReadTimeout is zero. c. ip_stUR20_1COM_CANopen_Param.udiWriteTimeout is zero. d. ip_stUR20_1COM_CANopen_Param.tTimeout is zero.
invalid process value	a. i_usiSlot < 1 or > 64 b. i_usiNodeId < 1 or > 127 c. i_uiIndex < 1 d. usiWriteDataLength = 0 or > 233
request timeout	The SDO read / write operation timed out. Please check communication settings.
read / write error	The sub-fb reports an error during the data write or read process in active state.
call error at request	The UR20-1COM-CANopen module detected an error in the asynchronous communication to the CANopen slave device.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_uiDeviceAdr	UINT	device slave address from Ethercat coupler
ip_stUR20_1COM_CANopen_Param	ST_UR20_1COM_CANopen_Param	this FB's parameters
i_usiSlot	USINT	hardware slot of the UR20 module within your UR20 station; 1 ...64
i_usiNodeId	USINT	Node ID of the CANopen slave device
i_uiIndex	UINT	device parameter index
i_usiSubIndex	USINT	device parameter sub-index
i_arusiData	ARRAY[0..232] OF USINT	data to be written to the device
i_usiWriteDataLength	USINT	data length to be written to the device

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

Structs

Name	Type	Comment
ST_UR20_1COM_CANopen_Param	STRUCT	Module Parameters
tTimeout	TIME	The general timeout for asynchronous SDO read / write operations. Default: TIME#50s0ms
udiReadTimeout	UDINT	The timeout in ms for the asynchronous SDO read FB. Default: 50000
udiWriteTimeout	UDINT	The timeout in ms for the asynchronous SDO write FB. Default: 50000

14.4 FB_UR20_1Com_CANopen_SDORRead_PN

Functional Description

The function block provides non-cyclic communication with a CANopen slave device via an UR20-1COM-CANopen module attached to a Profinet fieldbus coupler. This FB reads data from a CANopen slave device and transfers the data to your application.

The inputs of the function block must contain the reference of the Profinet coupler, the position of the CANopen module (slot), the node id and which parameter (index + subindex) shall be read from the slave device.

FB_UR20_1COM_CANopen_SDORRead_PN		
- i_xEnable <i>BOOL</i>		<i>BOOL</i> q_xStandby
- i_xExecute <i>BOOL</i>		<i>BOOL</i> q_xActive
- ip_PnDevice <i>REFERENCE TO IoDrvProfinet.IoDrvProfinet.PnSlaveDiag</i>		<i>BOOL</i> q_xBusy
- ip_stUR20_1COM_CANopen_Param <i>ST_UR20_1COM_CANopen_Param</i>		<i>BOOL</i> q_xError
- [i_usiSlot <i>USINT</i> := 1]		<i>ST_ErrorInfo</i> q_stError
- [i_usiNodeId <i>USINT</i> := 1]		<i>BOOL</i> q_xDone
- i_uiIndex <i>UINT</i>		<i>ARRAY[0..232] OF USINT</i> q_arusiData
- i_usiSubIndex <i>USINT</i>		<i>USINT</i> q_usiReadDataLength

Possible Errors

Reason	Description
invalid parameter	a. ip_PnDevice is not a valid reference. b. ip_stUR20_1COM_CANopen_Param.udiReadTimeout is zero.

Reason	Description
	c. ip_stUR20_1COM_CANopen_Param.udiWriteTimeout is zero. d. ip_stUR20_1COM_CANopen_Param.tTimeout is zero.
invalid process value	a. i_usiSlot < 1 or > 64 b. i_usiNodeId < 1 or > 127 c. i_uiIndex < 1
request timeout	The SDO read / write operation timed out. Please check communication settings.
read / write error	The sub-fb reports an error during the data write or read process in active state.
call error at request	The UR20-1COM-CANopen module detected an error in the asynchronous communication to the CANopen slave device.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_PnDevice	REFERENCE TO IoDrvProfinet.IoDrvProfinet.Pn SlaveDiag	Profinet device // fieldbus coupler
ip_stUR20_1COM_CANopen_Param	ST_UR20_1COM_CANopen_Param	this FB's parameters
i_usiSlot	USINT	hardware slot of the UR20 module within your UR20 station; 1 ...64
i_usiNodeId	USINT	Node ID of the CANopen slave device
i_uiIndex	UINT	device parameter index
i_usiSubIndex	USINT	device parameter sub-index

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end
q_arusiData	ARRAY [0..232] OF USINT	data from the CANopen device
q_usiReadDataLength	USINT	number of bytes read from the CANopen slave device

Structs

Name	Type	Comment
ST_UR20_1COM_CANopen_Param	STRUCT	Module Parameters
tTimeout	TIME	The general timeout for asynchronous SDO read / write operations. Default: TIME#50s0ms
udiReadTimeout	UDINT	The timeout in ms for the asynchronous SDO read FB. Default: 50000
udiWriteTimeout	UDINT	The timeout in ms for the asynchronous SDO write FB. Default: 50000

14.5 FB_UR20_1Com_CANopen_SDOWrite_PN**Functional Description**

The function block provides non-cyclic communication with a CANopen slave device via an UR20-1COM-CANopen module attached to a Profinet fieldbus coupler. This FB transfers write data from your application to a CANopen slave device.

The inputs of the function block must contain the reference of the Profinet coupler, the position of the CANopen module (slot), the node id and which parameter (index + subindex) shall be written.

FB_UR20_1COM_CANopen_SDOWrite_PN		
i_xEnable <i>BOOL</i>		<i>BOOL</i> q_xStandby
i_xExecute <i>BOOL</i>		<i>BOOL</i> q_xActive
ip_PnDevice <i>REFERENCE TO IoDrvProfinet.IoDrvProfinet.PnSlaveDiag</i>		<i>BOOL</i> q_xBusy
ip_stUR20_1COM_CANopen_Param <i>ST_UR20_1COM_CANopen_Param</i>		<i>BOOL</i> q_xError
[i_usiSlot <i>USINT</i> := 1]		<i>ST_ErrorInfo</i> q_stError
[i_usiNodeId <i>USINT</i> := 1]		<i>BOOL</i> q_xDone
i_uiIndex <i>UINT</i>		
i_usiSubIndex <i>USINT</i>		
i_arusiData <i>ARRAY[0..232] OF USINT</i>		
i_usiWriteDataLength <i>USINT</i>		

Possible Errors

Reason	Description
invalid parameter	a. ip_PnDevice is not a valid reference. b. ip_stUR20_1COM_CANopen_Param.udiReadTimeout is zero. c. ip_stUR20_1COM_CANopen_Param.udiWriteTimeout is zero. d. ip_stUR20_1COM_CANopen_Param.tTimeout is zero.
invalid process value	a. i_usiSlot < 1 or > 64 b. i_usiNodeId < 1 or > 127 c. i_uiIndex < 1 d. usiWriteDataLength = 0 or > 233
request timeout	The SDO read / write operation timed out. Please check communication settings.
read / write error	The sub-fb reports an error during the data write or read process in active state.
call error at request	The UR20-1COM-CANopen module detected an error in the asynchronous communication to the CANopen slave device.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_PnDevice	REFERENCE TO IoDrvProfinet.IoDrvProfinet.Pn SlaveDiag	Profinet device // fieldbus coupler
ip_stUR20_1COM_CANopen _Param	ST_UR20_1COM_CANopen_ Param	this FB's parameters
i_usiSlot	USINT	hardware slot of the UR20 module within your UR20 station; 1 ...64
i_usiNodeId	USINT	Node ID of the CANopen slave device
i_uiIndex	UINT	device parameter index
i_usiSubIndex	USINT	device parameter sub-index
i_arusiData	ARRAY[0..232] OF USINT	data to be written to the device
i_usiWriteDataLength	USINT	data length to be written to the device

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_xDone	BOOL	execution has come to its supposed end

Structs

Name	Type	Comment
ST_UR20_1COM_CANopen _Param	STRUCT	Module Parameters
tTimeout	TIME	The general timeout for asynchronous SDO read / write operations. Default: TIME#50s0ms
udiReadTimeout	UDINT	The timeout in ms for the asynchronous SDO read FB. Default: 50000
udiWriteTimeout	UDINT	The timeout in ms for the asynchronous SDO write FB. Default: 50000

15 libWiUr20ComCan

This library provides function blocks for the UR20-1COM-CAN module connected via a u-remote fieldbus coupler. These function blocks enable your application to read or write data from / to a CAN slave device.

Note: To receive CAN messages at the module it is necessary to activate and configure the RX Filter, RX Can ID and RX Mask parameter.

	RX Filter 1 ... 4	enabled (0) / disabled (1)	disabled
0	RX CAN ID 1 ... 4	0 ... 536870911	0
	RX Maske 1 ... 4	0 ... 536870911	536870911

e.g. to read all CAN messages on the fieldbus set RX Filter[1] to enabled, RX CAN ID[1] to 0 and RX Maske[1] to 0. Rx CAN id and Rx Mask are logically “**AND**” linked.

For the UR20-1COM-CAN module, the user can select different process input- and process output data sizes via the EDS file and a parametrization, see the UR20 handbook.

15.1 FB_UR20_1Com_CAN_Transparent

Functional Description

The function block provides an interface to the UR20-1Com-CAN module for the transparent mode. This function block configures and processes CAN communication between the Codesys application and one or more CAN devices connected to the UR20-1COM-CAN module.

The plc application on the one hand and the function block on the other hand exchange CAN messages via two arrays:

- The fixed-size array `iq_stCanTelegramTx` holds the CAN messages to be sent. The maximum size are 12 messages in one cycle for the 11bit ID mode and 9 messages for the 29 bit ID mode if the user has selected the maximum process data width of 122 bytes.
- The variable size array `iq_stCanTelegramRx` holds the received CAN messages. The size of the array must be larger than the maximally possible amount of CAN messages per cycle at the chosen process data size.

The function block expects and provides the hardware process data in two arrays of variable size: `iq_arbHwProcessInputData` and `iq_arbHwProcessOutputData`.

The process data width is variable and depends on the device description file and parametrization of the module. Weidmueller offers several device description files with different process data sizes. The decision which data width to use depends on the customer application. One 11bit ID CAN message takes 10 bytes of the process data and a 29bit ID CAN message takes 13 bytes of the process data.

Accordingly, if the user selects the maximum process image size of 122 bytes for input- and 122 bytes for output data, it is possible to read or send 12 messages per cycle.

After the user has enabled the FB by setting `i_xEnable` to **true**, the function block checks the sizes of the hardware process data arrays `iq_arbHwProcessInputData` and `iq_arbHwProcess-OutputData` and of the received CAN messages array `iq_arstCanTelegramRx`.

If any size check fails, the FB indicates this in the state **Error** and sets `q_xError` and additional error information in `q_stError`. If all size checks are ok, the FB transitions to the state **Standby**. On entry into the state **Standby**, the FB issues a Reset Module command to the UR20-1COM-CAN module. After the module has confirmed completion of the Reset Module command, the FB indicates its state **Standby** by setting `q_xStandby` to **true**. If any error occurs during the module reset or if the module reset times out, the FB enters its state **Error**, instead and provides detailed error information on its interface.

After the user has activated the `FB_UR20_1Com_CAN_Transparent` by setting `i_xExecute` to **true**, the FB will enter its state **Active** and begin sending telegrams.

Transmit Telegrams:

Use the variables `xTransmitMessage` and `xTransmitDone` in the struct `ST_UR20_1Com_CAN_CanTelegram` to control the data transfer:

- pass the CAN messages to be sent in the array `iq_stCanTelegramTx`
- set `xTransmitMessage` in `ST_UR20_1Com_CAN_CanTelegram` to send a message.
- The FB confirms sending of the message by setting `xTransmitDone` to **true**.

Receive telegrams:

The FB transfers CAN messages the UR20-1COM-CAN module has received into the array `iq_stCanTelegramRx`. The user sets the variable `i_xResetTelegramRxCnt` to reset the `q_diTelegramCounter`. This acknowledges that the user has processed the received data.

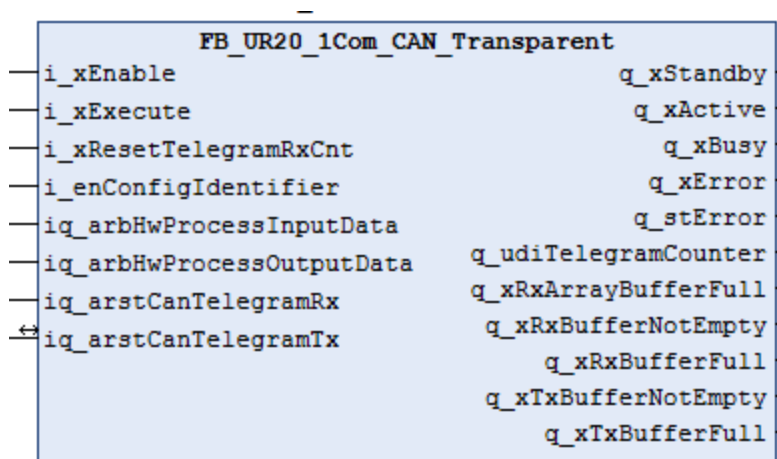
- the array `iq_stCanTelegramRx` contains the received telegrams.
- output `q_diTelegramCounter` shows the current number of received telegrams, it is also the current iterator of the input data array.
- the input `i_xResetTelegramRxCnt` resets the telegram counter `q_diTelegramCounter`.
- if the receive array `iq_stCanTelegramRx` is full, the FB indicates this by setting the output `q_diRxArrayBufferFull` to **true**. The FB cannot receive any more messages until the user sets `i_xResetTelegramRxCnt` to **true**.

The FB output `q_xRxBufferNotEmpty` indicates whether new data is available in the receive buffer of the module.

The FB output `q_xRxBufferFull` indicates if the receive buffer of the module is full. If this output is **true**, the UR20-1COM-CAN cannot store further new CAN messages.

To reset a function block error, it is necessary to reset the inputs `i_xExecute` and `i_xEnable`.

Please refer to the u-remote manual for further details.



Possible Errors

Reason	Description
Invalid Parameter	<p>Check the input parameter of the function block. The FB's parameters must be:</p> <ol style="list-style-type: none"> size of <code>iq_arbHwProcessInputData</code> = <code>iq_arbHwProcessOutputData</code> array size of <code>iq_arbHwProcessInputData</code> = [122],[62],[28] or [15] array size of <code>iq_arstCanTelegramRx</code> >= max amount of possible CAN msg per cycle. Calculate with 10 bytes per 11bit-ID msg. and 13 bytes per 29 bit-ID msg.
input data conversion	The input data is corrupt and cannot be processed further. Check the module's process data mapping. Check the detailed error information.
output data conversion	The output data is corrupt. Check the configured CAN messages and check the detailed error information.
reading and sending procedure timed out	During the reading and sending sequence, the module response timed out.
shutdown procedure timed out	During the shutdown sequence, the module response timed out.
module reset timeout	A module response timed out while the RX and TX buffers were flushed.

Inputs

Name	Type	Comment
<code>i_xEnable</code>	BOOL	enables the function block by switching from idle to standby
<code>i_xExecute</code>	BOOL	activates the function block by switching from standby to active
<code>i_xResetTelegramRxCnt</code>	BOOL	acknowledge that user has processed the received data
<code>i_enConfigIdentifier</code>	ET_UR20_1Com_CAN_Identifier	CAN identifier configured in the module hardware

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_udiTelegramCounter	UDINT	number of received telegrams
q_xRxArrayBufferFull	BOOL	true indicates that the user's receive array buffer is full
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

InOut

Name	Type	Comment
iq_arbHwProcessInputData	ARRAY [*] OF BYTE	Array of bytes for process data input from hardware module
iq_arbHwProcessOutputData	ARRAY [*] OF BYTE	Array of bytes for process data output to the hardware module
iq_arstCanTelegramRx	ARRAY [*] OF ST_UR20_1Com_CAN_CanTelegram	CAN telegrams received by the module
iq_stCanTelegramTx	ARRAY [0..11] OF ST_UR20_1Com_CAN_CanTelegram	CAN telegrams that shall be sent

Structs

Name	Type	Comment
ST_UR20_1Com_CAN_CanTelegram	STRUCT	CAN telegram data
uiCanIdentifier	UINT	CAN identifier for output data 0..2047
xCanRTR	BOOL	RTR bit for output data (Remote Transmission Req.)
usiDataLength	USINT	amount of user data that shall be sent in current message. Value range is 0..8 bytes
arbCanData	ARRAY [0..7] OF BYTE	array of user data, 8 bytes
xTransmitMessage	BOOL	if send flag is true the message will be transmitted
xTransmitDone	BOOL	true when the message has been sent

16 libWiUr20AiUi

The UR20-xAI-UI-xx-xxxx-xx modules feature two or four analog input channels. This library provides function blocks for parametrization of the UR20_xAI-UI-xx-xxxx-xx u-remote modules via the System Bus and structs and functions for Diagnostic Alarm message interpretation.

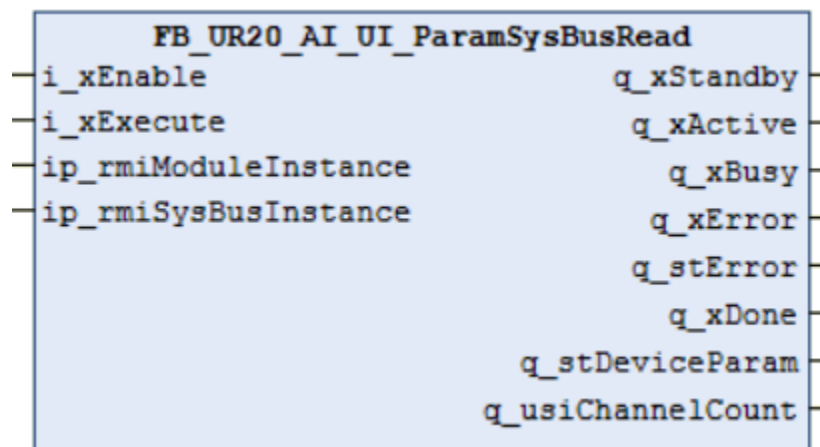
16.1 FB_UR20_AI_UI_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block shall be used in combination with the following U-Remote modules: UR20-2AI-UI-16, UR20-4AI-UI-12, UR20-4AI-UI-16 and UR20-4AI-UI-16-HD.

In state **active**, the function block outputs the detected number of channels on `q_usiChannelCount`.

Please observe that the output struct `q_stDeviceParam` holds parameters for modules with four channels. If you use this FB with a two-channel module, please ignore the slots two and three in the arrays in `q_stDeviceParam`.



Specific Outputs

Name	Type	Comment
<code>q_stDeviceParam</code>	ST_UR20_xAI_UI_DeviceParam	the device parameter set
<code>q_usiChannelCount</code>	USINT	the number of this module's input channels

Structs

Name	Type	Comment
ST_UR20_xAI_UI_DeviceParam	STRUCT	The members of this data type represent the parametrization of the UR20-xAI-UI-xx u-remote modules.

etFrequencySuppression	ET_UR20_AI_UI_FrequencySuppression	selects the signal frequency suppression
aretDataFormat	ARRAY [0..3] OF ET_UR20_AI_UI_DataFormat	selects the data format for the digital representation of the signals
aretMeasurementRange	ARRAY [0..3] OF ET_UR20_AI_UI_MeasurementRange	selects the measurement range

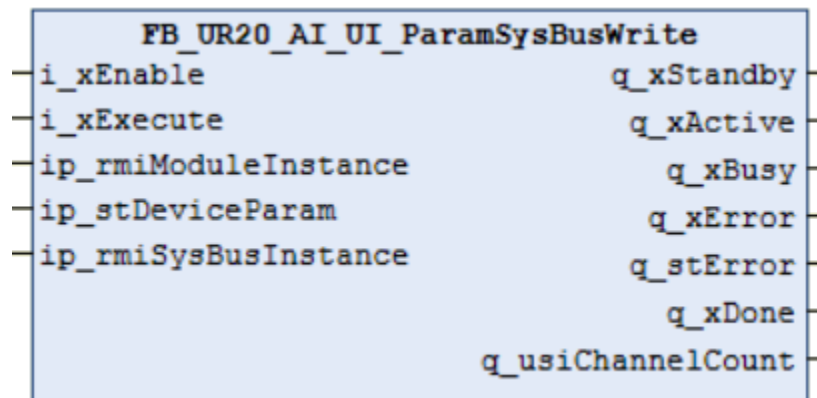
16.2 FB_UR20_AI_UI_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block can be used in combination with the following U-Remote modules: UR20-2AI-UI-16, UR20-4AI-UI-12, UR20-4AI-UI-16 and UR20-4AI-UI-16-HD.

In the state **active**, the FB derives the channel number from the module's TypeName and outputs the detected number of channels in q_usiChannelCount.

ip_stDeviceParam's data type ST_UR20_4AI_UI_DeviceParam represents modules with four channels. Please also use this datatype for the UR20-2AI-UI-12 and -16 and ignore the parameters for channels two and three.



Specific Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_xAI_UI_DeviceParam	the device parameter set to write

Specific Outputs

Name	Type	Comment
q_usiChannelCount	USINT	the number of this module's input channels

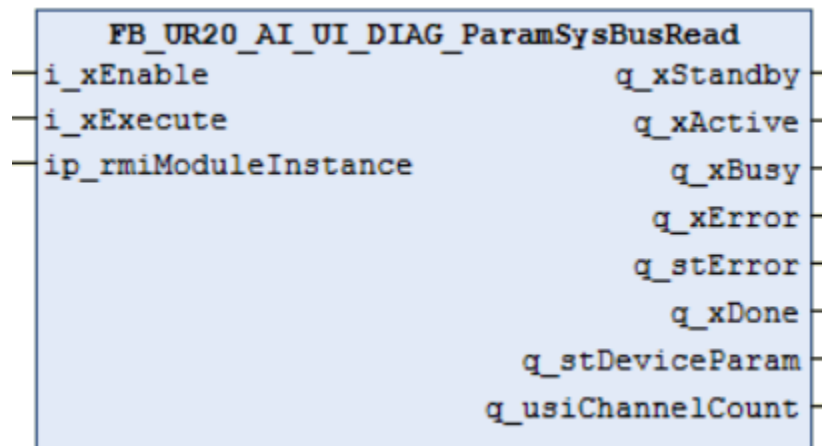
16.3 FB_UR20_AI_UI_DIAG_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block shall be used in combination with the following U-Remote modules: UR20-2AI-UI-16-DIAG, UR20-4AI-UI-16-DIAG and UR20-4AI-UI-16-DIAG-HD.

In state **active**, the FB outputs the detected number of channels on q_usiChannelCount.

Please observe that the output struct q_stDeviceParam holds parameters for modules with four channels. If you use this FB with a two-channel module, please ignore the slots two and three in the arrays in q_stDeviceParam.



Specific Outputs

Name	Type	Comment
q_stDeviceParam	ST_UR20_xAI_UI_DIAG_DeviceParam	the device parameter set
q_usiChannelCount	USINT	the number of this module's input channels

Structs

Name	Type	Comment
ST_UR20_xAI_UI_DIAG_DeviceParam	STRUCT	This data type extends ST_UR20_4AI_UI_DeviceParam with the additional members of a UR20-xAI-UI-16-DIAG-xx's parametrization.
etFrequencySuppression	ET_UR20_AI_UI_FrequencySuppression	selects the signal frequency suppression
aretDataFormat	ARRAY [0..3] OF ET_UR20_AI_UI_DataFormat	selects the data format for the digital representation of the signals
aretMeasurementRange	ARRAY [0..3] OF ET_UR20_AI_UI_MeasurementRange	selects the measurement range

aretChannelDiagnosis	ARRAY [0..3] OF ET_UR20_AI_UI_ChannelDiagnosis	selects diagnosis messages on / off for all channels
aretDiagShortCircuit	ARRAY [0..3] OF ET_UR20_AI_UI_DiagShortCircuit	selects sensor supply short circuit as diagnosis alarm trigger
aretDiagOpenWire	ARRAY [0..3] OF ET_UR20_AI_UI_DiagOpenWire	selects sensor supply open wire as diagnosis alarm trigger

Name	Type	Comment
ST_UR20_xAI_UI_DIAG_Device Param	STRUCT	The module-specific part of a UR20-xAI-UI-16-DIAG-xx module diagnostic message.
stErrorIndicator	ST_UR20_DiagDataErrorIndicator	Byte 0: error indicator
enModuleType	ET_UR20_ModuleType	Byte 1: default value 0x0F
bErrorByte2	BYTE	Byte 2: default value 0
stErrorByte3	ST_UR20_DiagDataErrorType	Byte 3: default value 0
enChannelType	ET_UR20_ChannelType	Byte 4: default value 0x70
bNumberOfDiagBits	BYTE	Byte 5: number of diagnostic bit per channel
bNumberOfChannels	BYTE	Byte 6: number of similar channels per module
stErrorChannel	ST_UR20_DiagDataErrorAtChannel	Byte 7: indicates which channel has the error
arbChannelError	ARRAY [0..2] OF BYTE	BYTE 8..10: expanded diagnosis
dwTimeStamp	DWORD	BYTE 43..46: time stamp µs
arst4AIUIDiagErrorChannel	ARRAY [0..3] OF ST_UR20_xAI_UI_DiagChannelError	Byte 11..14, channel 0..3 error diagnostic flags

Name	Type	Comment
ST_UR20_xAI_UI_DiagChannel Error	STRUCT	The channel error bits of a UR20-xAI-UI-16-DIAG-xx module.
xParameterError	BOOL	the module has received invalid parameters
xOverload	BOOL	an overload has occurred
xSensorSupplyLineBreak	BOOL	the 24V sensor supply has broken
xExternalShortCircuit	BOOL	the 24V sensor supply is shorted
xLineBreak	BOOL	the input signal is not in the selected measurement range
xLowerLimitExceeded	BOOL	the input signal has exceeded the lower limit
xUpperLimitExceeded	BOOL	the input signal has exceeded the upper limit

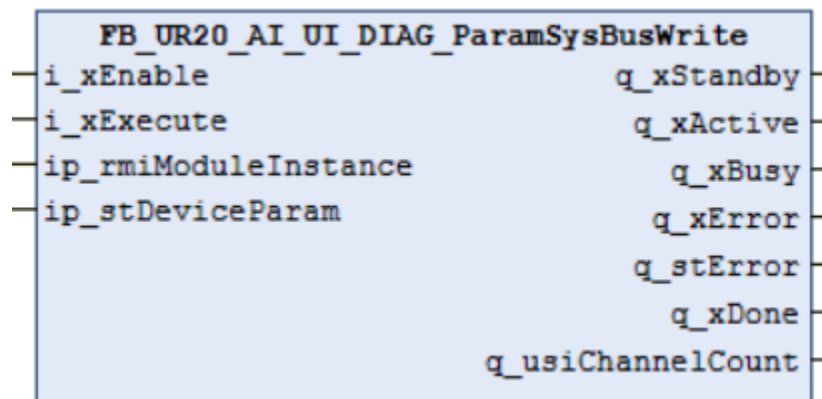
16.4 FB_UR20_AI_UI_DIAG_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block can be used in combination with the following U-Remote modules: UR20-2AI-UI-16-DIAG, UR20-4AI-UI-16-DIAG and UR20-4AI-UI-16-DIAG-HD.

In the state **active**, the FB outputs the detected number of channels on q_usiChannelCount.

ip_stDeviceParam's data type ST_UR20_4AI_UI_DIAG_DeviceParam represents modules with four channels. Please also use this datatype for the UR20-2AI-UI-16-DIAG and ignore the parameters for channels two and three.



Specific Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_xAI_UI_DIAG_DeviceParam	the device parameter set to write

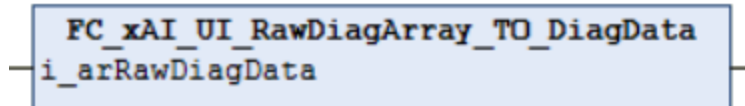
Specific Outputs

Name	Type	Comment
q_usiChannelCount	USINT	the number of this module's input channels

16.5 FC_xAI_UI_RawDiagArray_TO_DiagData

Please consult the Application Note AN0107v1-UC20-u-OS CODESYS Diagnosis- and Process Alarms for an overview over the alarm message transfer.

This function converts a raw diagnosis alarm message array to a more readable format. It is intended for diagnosis alarm messages of the UR20-xAI-UI-16-DIAG-xx u-remote modules. Please consult the u-remote manual for the individual UR20 modules' specifications.



Inputs

Name	Type	Comment
i_arRawDiagData	ARRAY [0..46] OF BYTE	raw diagnostic alarm message from UR20 module

Return

Name	Type	Comment
FC_xAI_UI_RawDiagArray_TO_DiagData	ST_UR20_xAI_UI_DiagData	The diagnostic alarm message in a struct.

Structs

Name	Type	Comment
ST_UR20_xAI_UI_DiagData	STRUCT	This struct represents the module-specific part of a diagnostic message of a UR20-xAI-UI-16-DIAG-xx module. Please find the common part in libWiUr20Diag -> ST_UR20_DiagData-UremoteClass.
stErrorIndicator	ST_UR20_DiagDataErrorIndicator	Byte 0: error indicator
enModuleType	ET_UR20_ModuleType	Byte 1: default value 0x0F
bErrorByte2	BYTE	Byte 2: default value 0
stErrorByte3	ST_UR20_DiagDataErrorType	Byte 3: default value 0
enChannelType	ET_UR20_ChannelType	Byte 4: default value 0x70
bNumberOfDiagBits	BYTE	Byte 5: number of diagnostic bit per channel
bNumberOfChannels	BYTE	Byte 6: number of similar channels per module
stErrorChannel	ST_UR20_DiagDataErrorAtChannel	Byte 7: indicates which channel has the error
arbChannelError	ARRAY [0..2] OF BYTE	BYTE 8..10: expanded diagnosis
dwTimeStamp	DWORD	BYTE 43..46: time stamp µs
arst4AIUIDiagErrorChannel	ARRAY [0..3] OF ST_UR20_xAI_UI_DiagChannelError	Byte 11..14, channel 0..3 error diagnostic flags

Name	Type	Comment
ST_UR20_xAI_UI_DiagChannelError	STRUCT	This struct represents the channel error bits of a UR20-xAI-UI-16-DIAG-xx module.
xParameterError	BOOL	the module has received invalid parameters
xOverload	BOOL	an overload has occurred
xSensorSupplyLineBreak	BOOL	the 24V sensor supply has broken
xExternalShortCircuit	BOOL	the 24V sensor supply is shorted
xLineBreak	BOOL	the input signal is not in the selected measurement range
xLowerLimitExceeded	BOOL	the input signal has exceeded the lower limit
xUpperLimitExceeded	BOOL	the input signal has exceeded the upper limit

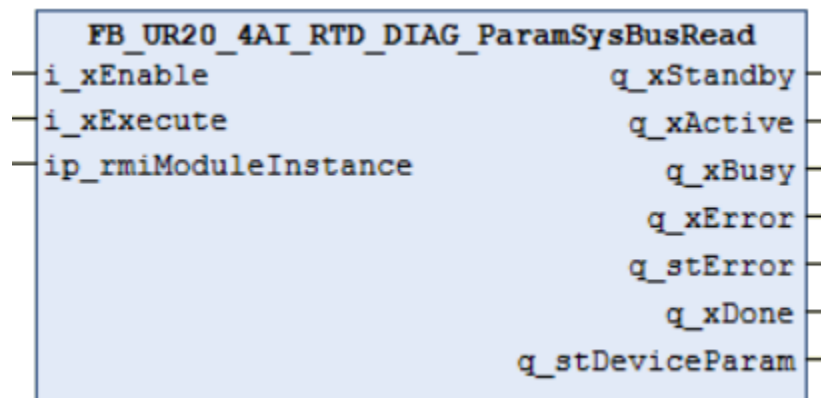
17 libWiUr20RTD

The UR20-4AI-RTD-DIAG module features four resistive temperature device input channels. This library provides function blocks for parametrization of the UR20-4AI-RTD-xx-DIAG u-remote modules via the System Bus and structs and functions for Diagnostic- and Process Alarm message interpretation.

17.1 FB_UR20_4AI_RTD_DIAG_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block shall be used in combination with the following U-Remote modules: UR20-4AI-RTD-DIAG.



Specific Outputs

Name	Type	Comment
q_stDeviceParam	ST_UR20_4AI_RTD_DIAG_DeviceParam	the device parameter set

Structs

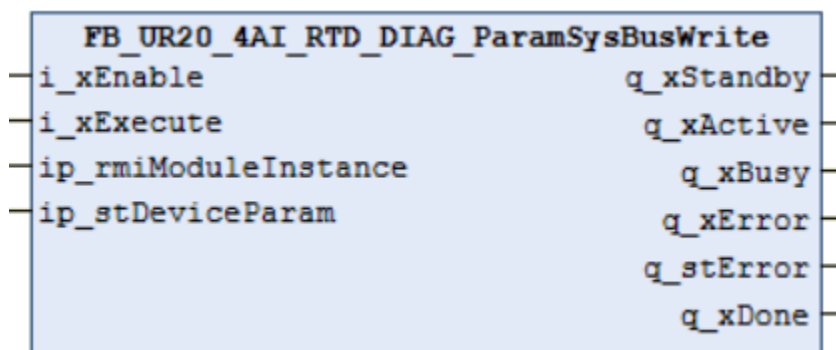
Name	Type	Comment
ST_UR20_4AI_RTD_DIAG_DeviceParam	STRUCT	This data type represents the UR20-4AI-RTD-DIAG module's parameter set.
etTemperatureUnit	ET_UR20_4AI_RTD_TemperatureUnit	the temperature unit for the measurement values of all channels
aretConnectionType	ARRAY [0..3] OF ET_UR20_4AI_RTD_ConnectionType	selects the sensor connection type for each channel
aretConversionTime	ARRAY [0..3] OF ET_UR20_4AI_RTD_ConversionTime	selects the conversion time settings for each channel
aretChannelDiagnosis	ARRAY [0..3] OF ET_UR20_4AI_RTD_OnOff	selects diagnosis messages on / off for each channel
aretLimitValueMonitoring	ARRAY [0..3] OF ET_UR20_4AI_RTD_OnOff	selects limit value monitoring on / off for each channel

ariHighLimitValue	ARRAY [0..3] OF INT	high limit value
ariLowLimitValue	ARRAY [0..3] OF INT	low limit value
aretMeasurementRange	ARRAY [0..3] OF ET_UR20_4AI_RTD_MeasurementRange	selects the measurement range for each channel

17.2 FB_UR20_4AI_RTD_DIAG_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block can be used in combination with the following U-Remote modules: UR20-4AI-RTD-DIAG.



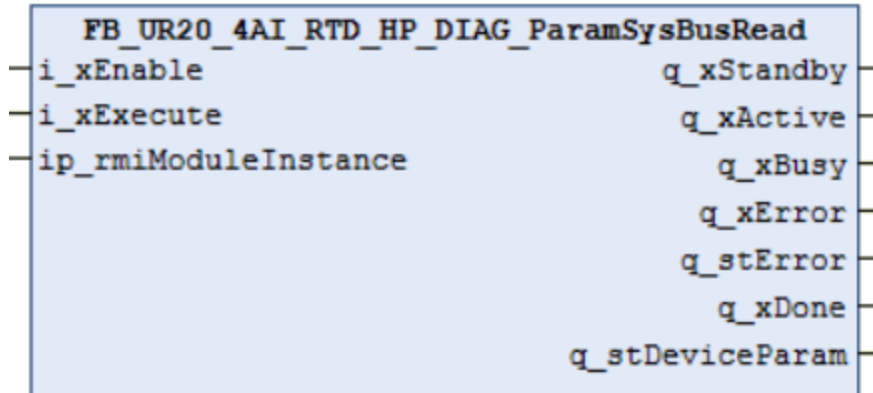
Specific Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_4AI_RTD_DIAG_DeviceParam	device parameters to write

17.3 FB_UR20_4AI_RTD_HP_DIAG_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block shall be used in combination with the following U-Remote modules: UR20-4AI-RTD-HP-DIAG.



Outputs

Name	Type	Comment
q_stDeviceParam	ST_UR20_4AI_RTD_HP_DI AG_DeviceParam	the device parameter set

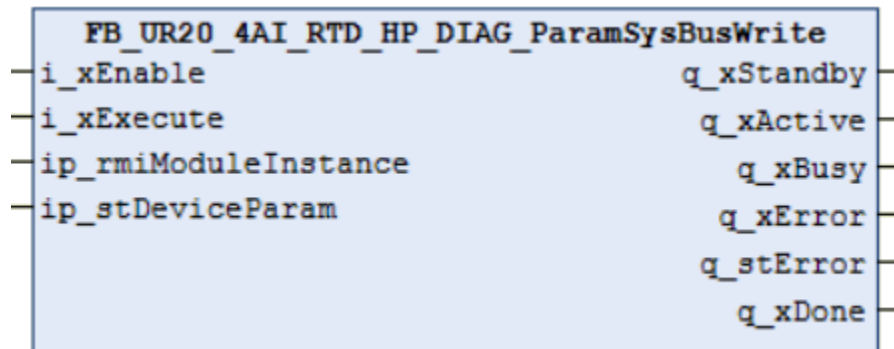
Structs

Name	Type	Comment
ST_UR20_4AI_RTD_HP_DIAG_ DeviceParam	STRUCT	This data type represents the UR20-4AI-RTD- HP-DIAG module's parameter set.
etTemperatureUnit	ET_UR20_4AI_RTD_Te mperatureUnit	the temperature unit for the measurement values of all channels
aretConnectionType	ARRAY [0..3] OF ET_UR20_4AI_RTD_ ConnectionType	selects the sensor connection type for each channel
aretConversionTime	ARRAY [0..3] OF ET_UR20_4AI_RTD_ ConversionTime	selects the conversion time settings for each channel
aretChannelDiagnosis	ARRAY [0..3] OF ET_UR20_4AI_RTD_ OnOff	selects diagnosis messages on / off for each channel
aretLimitValueMonitoring	ARRAY [0..3] OF ET_UR20_4AI_RTD_ OnOff	selects limit value monitoring on / off for each channel
ariHighLimitValue	ARRAY [0..3] OF INT	high limit value
ariLowLimitValue	ARRAY [0..3] OF INT	low limit value
aretMeasurementRange	ARRAY [0..3] OF ET_UR20_4AI_RTD_ MeasurementRange	selects the measurement range for each channel
aretUserCalibration	ARRAY [0..3] OF ET_UR20_4AI_RTD_ OnOff	enable or disable user calibration for each channel
ariUserCalibrationOffset	ARRAY [0..3] OF INT	the user calibration offsets for each channel
aruiUserCalibrationGain	ARRAY [0..3] OF UINT	the user calibration gains for each channel

17.4 FB_UR20_4AI_RTD_HP_DIAG_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block can be used in combination with the following U-Remote modules: UR20-4AI-RTD-HP-DIAG.



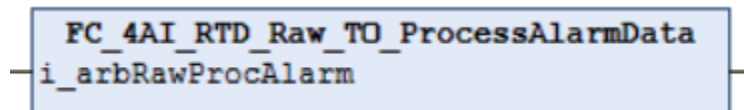
Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_4AI_RTD_HP_DIAG_DeviceParam	device parameters to write

17.5 FC_4AI_RTD_Raw_TO_ProcessAlarmData

Please consult the Application Note AN0107v1-UC20-u-OS CODESYS Diagnosis- and Process Alarms for an overview over the alarm message transfer.

This function converts a raw process alarm message array to a more readable format. Please also consult the u-remote manual for the individual UR20 modules' specifications.



Inputs

Name	Type	Comment
i_arbRawProcAlarm	ARRAY [0..3] OF BYTE	the raw process alarm message

Return

Name	Type	Comment
FC_4AI_RTD_Raw_TO_ProcessAlarmData	ST_UR20_4AI_RTD_DIAG_ProcAlarmData	the process alarm message in a module-specific struct

Structs

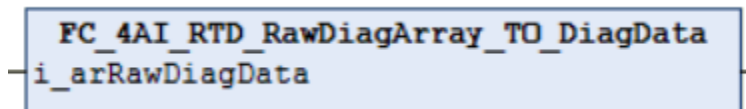
Name	Type	Comment
ST_UR20_4AI_RTD_DIAG_ProcAlarmData	STRUCT	This struct represents a UR20-4AI-TC-DIAG process alarm message in a more readable format.
stUpperLimitAlarm	ST_UR20_ProcAlarmAtChannel	Byte 0: Upper Limit Alarm. Each bit represents one channel of the module.
stLowerLimitAlarm	ST_UR20_ProcAlarmAtChannel	Byte 1: Lower Limit Alarm. Each bit represents one channel of the module.
wTimestamp	WORD	BYTE 2..3, the two lowest byte of the internal 32-bit time stamp in μ s

Name	Type	Comment
ST_UR20_ProcAlarmAtChannel	STRUCT	A high or low process alarm occurred at the channel(s) with the set bit(s).
xChannel0	BOOL	channel 0
xChannel1	BOOL	channel 1
xChannel2	BOOL	channel 2
xChannel3	BOOL	channel 3

17.6 FC_4AI_RTD_RawDiagArray_TO_DiagData

Please consult the Application Note AN0107v1-UC20-u-OS CODESYS Diagnosis- and Process Alarms for an overview over the alarm message transfer.

This function converts a raw diagnosis alarm message array to a more readable format. Please consult the u-remote manual for the individual UR20 modules' specifications.

**Inputs**

Name	Type	Comment
i_arRawDiagData	ARRAY [0..46] OF BYTE	raw diagnostic alarm message from UR20 module

Return

Name	Type	Comment
FC_4AI_RTD_RawDiagArray_TO_DiagData	ST_UR20_4AI_RTD_DiagData	The diagnostic alarm message in a struct.

Structs

Name	Type	Comment
ST_UR20_4AI_RTD_DiagData	STRUCT	This struct represents the module-specific part of a diagnostic message of a UR20-4AI-RTD-xx-DIAG module. Please find the common part in libWiUr20Diag -> ST_UR20_DiagDataUremoteClass.
stErrorIndicator	ST_UR20_DiagDataErrorIndicator	Byte 0: error indicator
enModuleType	ET_UR20_ModuleType	Byte 1: default value 0x0F
bErrorByte2	BYTE	Byte 2: default value 0
stErrorByte3	ST_UR20_DiagDataErrorType	Byte 3: default value 0
enChannelType	ET_UR20_ChannelType	Byte 4: default value 0x70
bNumberOfDiagBits	BYTE	Byte 5: number of diagnostic bit per channel
bNumberOfChannels	BYTE	Byte 6: number of similar channels per module
stErrorChannel	ST_UR20_DiagDataErrorAtChannel	Byte 7: indicates which channel has the error
arbChannelError	ARRAY [0..2] OF BYTE	BYTE 8..10: expanded diagnosis
dwTimeStamp	DWORD	BYTE 43..46: time stamp µs
arst4AIRTDDiagErrorChannel	ARRAY [0..3] OF ST_UR20_4AI_RTD_DiagChannelError	Byte 11..14, channel 0..3 error diagnostic flags

Name	Type	Comment
ST_UR20_4AI_RTD_DiagChannelError	STRUCT	This struct represents the channel error bits of a UR20-4AI-RTD-xx-DIAG module.
xParameterError	BOOL	the module has received invalid parameters
xReserved	BOOL	reserved
xReserved1	BOOL	reserved
xReserved2	BOOL	reserved
xLineBreak	BOOL	the module has detected a line break
xProcessAlarmLost	BOOL	the module's process alarm buffer has overflowed
xLowerLimitExceeded	BOOL	the input signal has exceeded the lower limit
xUpperLimitExceeded	BOOL	the input signal has exceeded the upper limit

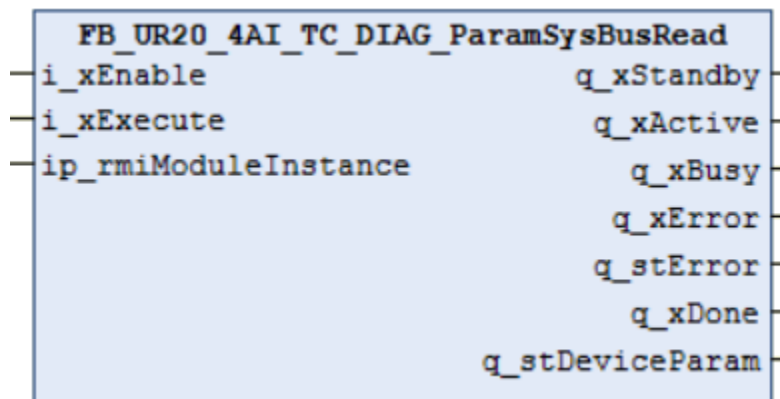
18 libWiUr20TC

The UR20-4AI-TC-DIAG module features four thermocouple input channels. This library provides function blocks for parametrization of the UR20-4AI-TC-DIAG u-remote module via the system bus and structs and functions for diagnostic- and process alarm message interpretation.

18.1 FB_UR20_4AI_TC_DIAG_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block is intended for use with the UR20-4AI-TC-DIAG U-Remote module.



Specific Outputs

Name	Type	Comment
q_stDeviceParam	ST_UR20_4AI_TC_DIAG_DeviceParam	the device parameter set

Structs

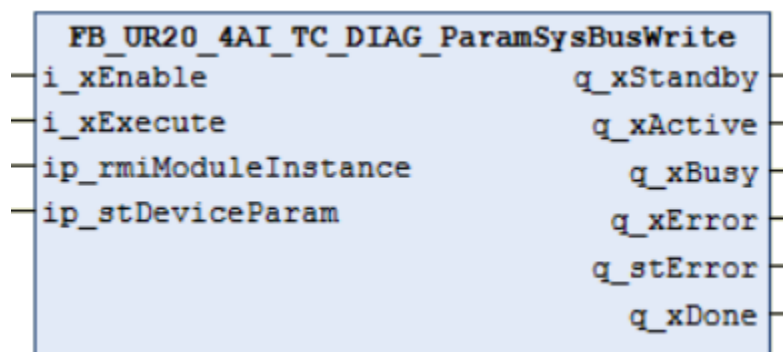
Name	Type	Comment
ST_UR20_4AI_TC_DIAG_DeviceParam	STRUCT	This data type represents the UR20-4AI-TC-DIAG module's parameter set.
etTemperatureUnit	ET_UR20_4AI_TC_TemperatureUnit	the temperature unit for the measurement values of all channels
aretMeasurementRange	ARRAY [0..3] OF ET_UR20_4AI_TC_MeasurementRange	selects the measurement range for each channel
aretColdJunctionCompensation	ARRAY [0..3] OF ET_UR20_4AI_TC_CJCRange	selects the cold junction compensation for each channel
aretConversionTime	ARRAY [0..3] OF ET_UR20_4AI_TC_ConversionTime	selects the conversion time settings for each channel
aretChannelDiagnosis	ARRAY [0..3] OF ET_UR20_4AI_TC_OnOff	selects diagnosis messages on / off for each channel

aretLimitValueMonitoring	ARRAY [0..3] OF ET_UR20_4AI_TC_OnOff	selects limit value monitoring on / off for each channel
ariHighLimitValue	ARRAY [0..3] OF INT	high limit value
ariLowLimitValue	ARRAY [0..3] OF INT	low limit value

18.2 FB_UR20_4AI_TC_DIAG_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block is intended for use with the UR20-4AI-TC-DIAG U-Remote module.



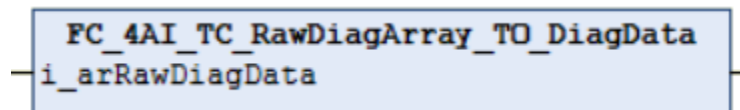
Specific Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_4AI_TC_DIAG_DeviceParam	device parameters to write

18.3 FC_4AI_TC_RawDiagArray_TO_DiagData

Please consult the Application Note AN0107v1-UC20-u-OS CODESYS Diagnosis- and Process Alarms for an overview over the alarm message transfer.

This function converts a raw diagnosis alarm message array to a more readable format.



Inputs

Name	Type	Comment
i_arRawDiagData	ARRAY [0..46] OF BYTE	raw diagnostic alarm message from UR20 module

Return

Name	Type	Comment
FC_4AI_TC_RawDiagArray_TO_DiagData	ST_UR20_4AI_TC_DiagData	The diagnostic alarm message in a struct.

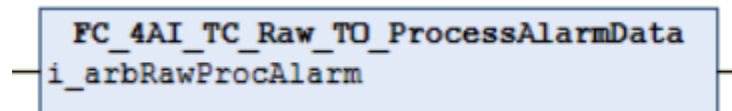
Structs

Name	Type	Comment
ST_UR20_4AI_TC_DiagData	STRUCT	This struct represents the module-specific part of a diagnostic message of a UR20-4AI-TC-DIAG module. Please find the common part in libWiUr20Diag -> ST_UR20_DiagDataUremoteClass.
stErrorIndicator	ST_UR20_DiagDataErrorIndicator	Byte 0: error indicator
enModuleType	ET_UR20_ModuleType	Byte 1: default value 0x0F
bErrorByte2	BYTE	Byte 2: default value 0
stErrorByte3	ST_UR20_DiagDataErrorType	Byte 3: default value 0
enChannelType	ET_UR20_ChannelType	Byte 4: default value 0x70
bNumberOfDiagBits	BYTE	Byte 5: number of diagnostic bit per channel
bNumberOfChannels	BYTE	Byte 6: number of similar channels per module
stErrorChannel	ST_UR20_DiagDataErrorAtChannel	Byte 7: indicates which channel has the error
arbChannelError	ARRAY [0..2] OF BYTE	BYTE 8..10: expanded diagnosis
dwTimeStamp	DWORD	BYTE 43..46: time stamp µs
arst4AITCdiagErrorChannel	ARRAY [0..3] OF ST_UR20_4AI_TC_DiagChannelError	Byte 11..14, channel 0..3 error diagnostic flags

18.4 FC_4AI_TC_Raw_TO_ProcessAlarmData

Please consult the Application Note AN0107v1-UC20-u-OS CODESYS Diagnosis- and Process Alarms for an overview over the alarm message transfer.

This function converts a raw process alarm message array to a more readable format.

**Inputs**

Name	Type	Comment
i_arbRawProcAlarm	ARRAY [0..3] OF BYTE	the raw process alarm message

Return

Name	Type	Comment
FC_4AI_TC_Raw_T0_ProcessAlarmData	ST_UR20_4AI_TC_DIAG_ProcAlarmData	the process alarm message in a module-specific struct

Structs

Name	Type	Comment
ST_UR20_4AI_TC_DIAG_ProcAlarmData	STRUCT	This struct represents a UR20-4AI-TC-DIAG process alarm message in a more readable format.
stUpperLimitAlarm	ST_UR20_ProcAlarmAtChannel	Byte 0: Upper Limit Alarm. Each bit represents one channel of the module.
stLowerLimitAlarm	ST_UR20_ProcAlarmAtChannel	Byte 1: Lower Limit Alarm. Each bit represents one channel of the module.
wTimestamp	WORD	BYTE 2..3, the two lowest byte of the internal 32-bit time stamp in μ s

19 libWiUr20PWM

This library provides parametrization- and operation function blocks for the UR20-2PWM-PN-0.5A, UR20-2PWM-PN-2A and UR20-2PWM-I-2.5A-2DI-P modules. Please also refer to the UR20 handbook on Weidmüller's website for more detailed information on the modules.

19.1 FB_UR20_2PWM_PN_xA_Control

This function block derives from the LevelControlledBehavior model and provides control for the channels of a UR20_2PWM_PN_xA module: It calculates a module parametrization from its control parameter set and writes the module parametrization to the UR20_2PWM_PN_xA module. It accepts a duty cycle value for each channel in `i_ar1rDutyCycle` and calculates the correct pulse duration. It then passes the calculated pulse duration to the UR20_2PWM_PN_xA module and activates or deactivates the PWM channels according to the values at its input `i_arxChannelOutputEnable`.

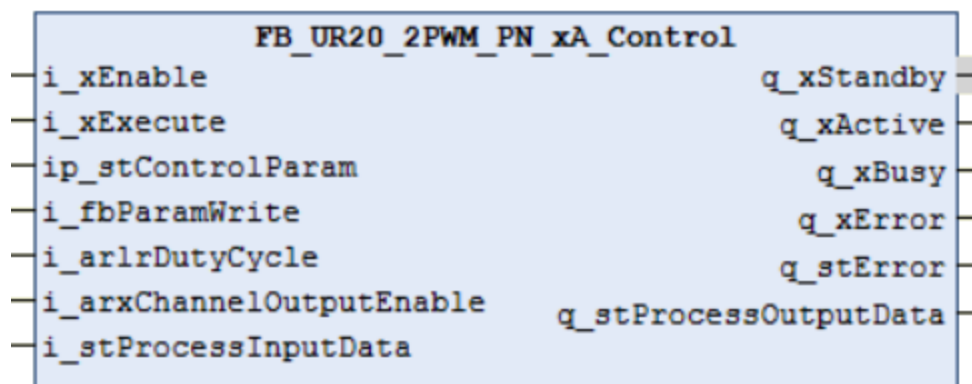
To do so, the user passes a reference to the correct `Param<fieldbus>Write()` function block instance to `i_fbParamWrite`. After the user has enabled `FB_UR20_2PWM_PN_xA_Control`, the function block checks its control parameter set. Then it shadow-copies the control parameter set, calculates the period durations and writes them to the UR20_2PWM_PN_xA module. Then the function block enters its state **standby**. The user activates `FB_UR20_2PWM_PN_xA_Control`. The function block becomes **active**, sets the desired output functions and continuously evaluates the desired duty cycle inputs `i_ar1rDutyCycle`. It sets the pulse durations according to the duty cycles and monitors its `i_arxChannelOutputEnable` inputs and enables or disables the UR20_2PWM_PN_xA module's PWM channel outputs, accordingly.

After the user has deactivated `FB_UR20_2PWM_PN_xA_Control`, the function block disables both PWM channels of the UR20_2PWM_PN_xA module and returns to the state **standby**.

If the initial control parameter check or the module parameter write fail, `FB_UR20_2PWM_PN_xA_Control` enters its state **error**.

A note regarding `i_fbParamWrite`: Instantiate a `FB_UR20_2PWM_PN_xA_<fieldbus>Write` function block suitable for your application. Then pass a reference to this instance to `i_ifParamWrite`. Here is an example:

```
fbParamSysbusWrite : FB_UR20_2PWM_PN_xA_ParamSysBusWrite;
fbPWMControl : FB_UR20_2PWM_PN_xA_Control;
fbPWMControl(i_fbParamWrite REF= fbParamSysbusWrite);
```



Possible Errors

Reason	Description
invalid module reference	invalid input parameter "ip_rmiModuleInstance"
i_fbParamWrite parameter error	i_fbParamWrite is not a valid reference to an instance of FB_UR20_2PWM_PN_xA_ParamAnyWrite.
i_fbParamWrite instance error	The FB_UR20_2PWM_PN_xA_Control forwards errors reported by the FB instance connected to i_fbParamWrite.
control parameter invalid	ip_stControlParam.arlrFrequency[0..1] is out of range. The allowed range is: 5,723 Hz <= f <= 39.939,834 Hz
process input value invalid	duty cycle is out of limits. The allowed range is: 0.0 <= duty cycle <= 1.0
channel wrongly on or off	One or both PWM channels are on when they shall be off or off when they shall be on, for more than 2 PLC cycles. Check the process input- and output data mapping to the module.

Inputs

Name	Type	Comment
i_xEnable	BOOL	enables the function block by switching from idle to standby
i_xExecute	BOOL	activates the function block by switching from standby to active
ip_stControlParam	ST_UR20_2PWM_PN_xA_ControlParam	parameters for operation of this function block
i_fbParamWrite	REFERENCE TO FB_UR20_2PWM_PN_xA_ParamAnyWrite	reference to a device parameter write function block
i_arlrDutyCycle	ARRAY [0..1] OF LREAL	the target duty cycles for the two PWM channels, in the range 0.0 .. 1.0
i_arxChannelOutputEnable	ARRAY [0..1] OF BOOL	true enables the PWM output
i_stProcessInputData	ST_UR20_2PWM_PN_xA_ProcessInputs	input for process data _from_ the hardware _to_ the plc

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_xError	BOOL	function block is in error state
q_stError	ST_ErrorInfo	detailed error information
q_stProcessOutputData	ST_UR20_2PWM_PN_xA_ProcessOutputs	output for process data from the plc to the hardware

Structs

Name	Type	Comment
ST_UR20_2PWM_PN_xA_ControlParam	STRUCT	This struct holds the control parameters for a UR20_2PWM_PN_xA module.
arLrFrequency	ARRAY [0..1] OF LREAL	target frequency of the PWM signal for each channel
aretOutputFunction	ARRAY [0..1] OF ET_UR20_2PWM_PN_xA_OutputFunction	output function selection: P- vs. PN-switching

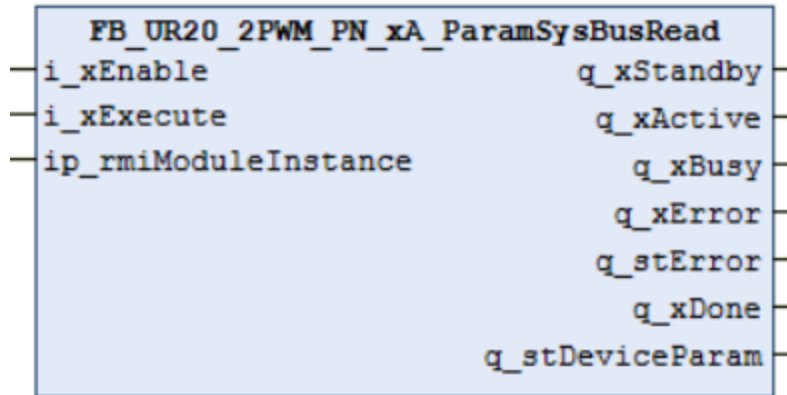
Name	Type	Comment
ST_UR20_2PWM_PN_xA_ProcessInputs	STRUCT	This struct holds the process input values for a UR20-2PWM-PN-xA module.
wStatusWord0	WORD	channel 0's status word
wStatusWord1	WORD	channel 1's status word

Name	Type	Comment
ST_UR20_2PWM_PN_xA_ProcessOutputs	STRUCT	This struct holds the process output values for a UR20-2PWM-PN-xA module.
dwPulseDuration0	DWORD	pulse duration of channel 0
dwPulseDuration1	DWORD	pulse duration of channel 1
wControlWord0	WORD	channel 0's control word
wControlWord1	WORD	channel 1's control word

19.2 FB_UR20_2PWM_PN_xA_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block is intended for use with the UR20-2PWM-PN-0.5A and UR20-2PWM-PN-2A U-Remote modules.



Specific Outputs

Name	Type	Comment
q_stDeviceParam	ST_UR20_2PWM_PN_xA_DeviceParam	device parameters for the UR20_2PWM_PN_xA modules

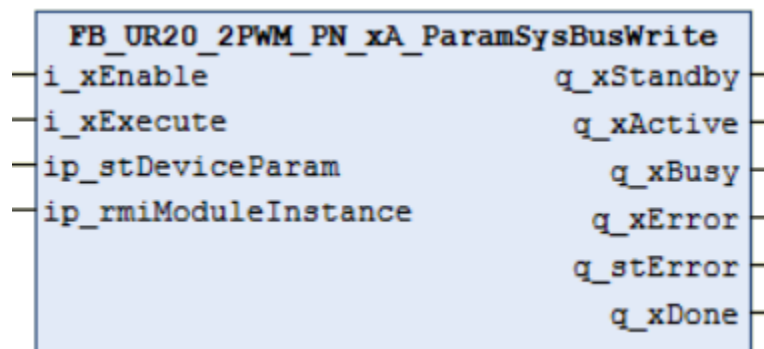
Structs

Name	Type	Comment
ST_UR20_2PWM_PN_xA_DeviceParam	STRUCT	This struct holds the device parameters for a UR20_2PWM_PN_xA module.
aruxiPeriodDuration	ARRAY [0..1] OF __UXINT	target frequency of the PWM signal for each channel

19.3 FB_UR20_2PWM_PN_xA_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block is intended for use with the UR20-2PWM-PN-xA U-Remote module.



Specific Possible Errors

Reason	Description
invalid process value	a. ip_stDeviceParam.aruxiPeriodDuration[] < 1202 b. ip_stDeviceParam.aruxiPeriodDuration[] > 8388607 Use a value >= 1202 and <= 8388607.

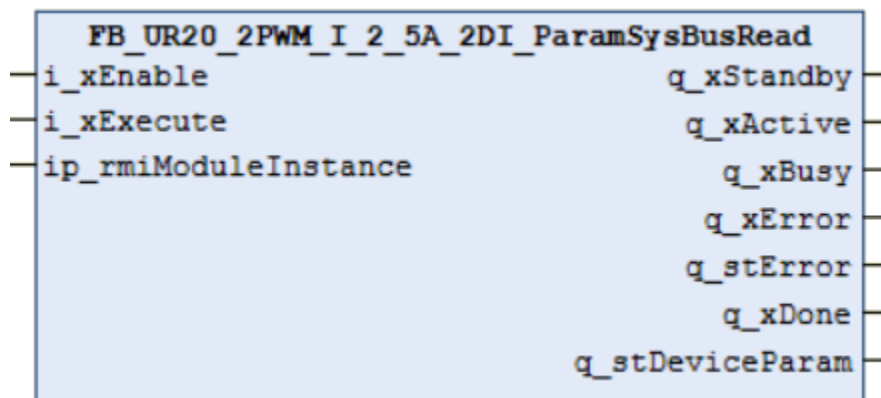
Specific Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_2PWM_PN_xA_DeviceParam	device parameters for the UR20_2PWM_PN_xA modules

19.4 FB_UR20_2PWM_I_2_5A_2DI_ParamSysBusRead

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block is intended for use with the UR20-2PWM-I-2.5A-2DI-P U-Remote module.

**Specific Outputs**

Name	Type	Comment
q_stDeviceParam	ST_UR20_2PWM_I_2_5A_2DI_DeviceParam	device parameters for the UR20_2PWM_PN_xA modules

Structs

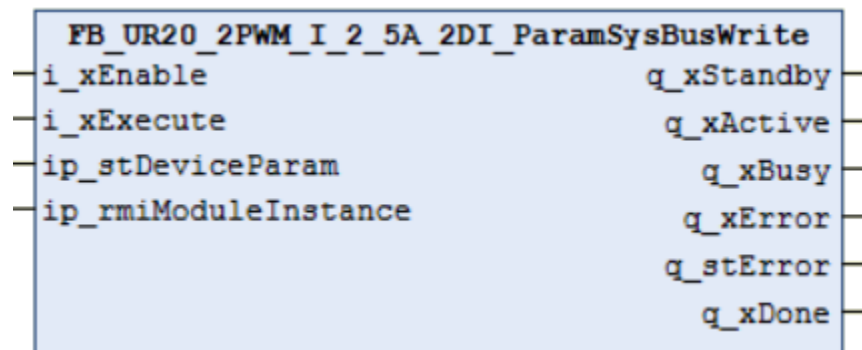
Name	Type	Comment
ST_UR20_2PWM_I_2_5A_2DI_DeviceParam	STRUCT	This struct holds the device parameters for a UR20_2PWM_I_2_5A_2DI module.
aruxiPeriodDuration	ARRAY [0..1] OF __UXINT	target frequency of the PWM signal for each channel
etFunctionDI0	ET_UR20_2PWM_I_2_5A_2DI_DI0function	function of the digital input

etInputDelay0	ET_UR20_DI_InputDelay	digital input delay
etChannelDiagnosis0	ET_UR20_OnOff	enable / disable channel diagnosis
etFunctionDI1	ET_UR20_2PWM_I_2_5A_2DI_DI1function	function of the digital input
etInputDelay1	ET_UR20_DI_InputDelay	digital input delay
etChannelDiagnosis1	ET_UR20_OnOff	enable / disable channel diagnosis
arstPWMout	ARRAY [2..3] OF ST_UR20_2PWM_I_2_5A_2DI_PWMParam	parameters for channels 2 and 3, current-controlled digital PWM outputs

19.5 FB_UR20_2PWM_I_2_5A_2DI_ParamSysBusWrite

Please refer to chapter 4 for the functional description of this FB and its input- and output and possible error lists.

The function block is intended for use with the UR20-2PWM-I-2.5A-2DI-P U-Remote module.



Specific Possible Errors

Reason	Description
invalid process value	a. <code>ip_stDeviceParam.arstPWMout[].uiMaxOutputCurrent</code> ≤ 0 or <code>ip_stDeviceParam.arstPWMout[].uiMaxOutputCurrent</code> > 3000
	b. <code>ip_stDeviceParam.arstPWMout[].uiFullScaleValue</code> ≤ 0 or <code>ip_stDeviceParam.arstPWMout[].uiFullScaleValue</code> > 32767
	c. <code>ip_stDeviceParam.arstPWMout[].iSubstituteValue</code> ≤ -32768
	d. <code>ip_stDeviceParam.arstPWMout[].uiDitheringFrequency</code> < 10 or <code>ip_stDeviceParam.arstPWMout[].uiDitheringFrequency</code> > 500
	e. <code>ip_stDeviceParam.arstPWMout[].uiDitheringAmplitude</code> < 1 or <code>ip_stDeviceParam.arstPWMout[].uiDitheringAmplitude</code> > 500
	f. <code>ip_stDeviceParam.arstPWMout[].uiDitheringSwitchOffRamp</code> > 32767
	g. <code>ip_stDeviceParam.arstPWMout[].uiValveCharacteristicY0</code> > 32767
	h. <code>ip_stDeviceParam.arstPWMout[].uiValveCharacteristicX1</code> > 32767
	i. <code>ip_stDeviceParam.arstPWMout[].uiValveCharacteristicY1</code> > 32767
	j. <code>ip_stDeviceParam.arstPWMout[].uiValveCharacteristicX2</code> > 32767

Reason	Description
	k. ip_stDeviceParam.arstPWMout[].uiValveCharacteristicY2 > 32767 l. ip_stDeviceParam.arstPWMout[].uiCurrentSwitchOnRamp > 32767 m. ip_stDeviceParam.arstPWMout[].uiCurrentSwitchOffRamp > 32767 n. ip_stDeviceParam.arstPWMout[].uiIntegratorConstant < 1 o. ip_stDeviceParam.arstPWMout[].uiProportionalConstant < 1

Specific Inputs

Name	Type	Comment
ip_stDeviceParam	ST_UR20_2PWM_I_2_5A_2DI_DeviceParam	device parameters for the UR20_2PWM_I_2_5A modules