EcoStruxure™
Innovation At Every Level

How to use Data_Exch for Modbus communication

Knowledge Base

Balaji Tekkiam

Rev 1.0
23 May 2021

Life Is On Schneider Electric

Disclaimer

⚠ CAUTION

This technical document is NOT a training or instruction tool or a substitute for hands-on training for the installation, use or maintenance of Schneider Electric equipment. It is provided merely as a demonstration of the installation and configuration of our products, and may ONLY be used as a supplemental reference by a competent service technician who is fully trained in the care, operation and maintenance of the installed electrical equipment. Although reasonable care has been taken to provide accurate and authoritative information, Schneider Electric assumes no responsibility for any consequences arising out of the use of this presentation.

Schneider Electric cautions you that electrical equipment should be installed, serviced or maintained ONLY by qualified electrical maintenance personnel. Remember that all sources of electrical power must be disconnected prior to visual inspection, tests, maintenance or installation of any electrical equipment. Failure to do so can result in serious personal injury, property damage, OR death.

Trademarks

Schneider Electric has made every effort to supply trademark information about company names, products and services mentioned in this manual. Trademarks shown below were derived from various sources.

EcoStruxure Control Expert, Telemecanique Premium, Modicon Quantum, Modicon M340, Modicon M580 ePAC are trademarks owned by Schneider Electric or its affiliated companies. All other trademarks are the property of their respective owners.

General Notice:

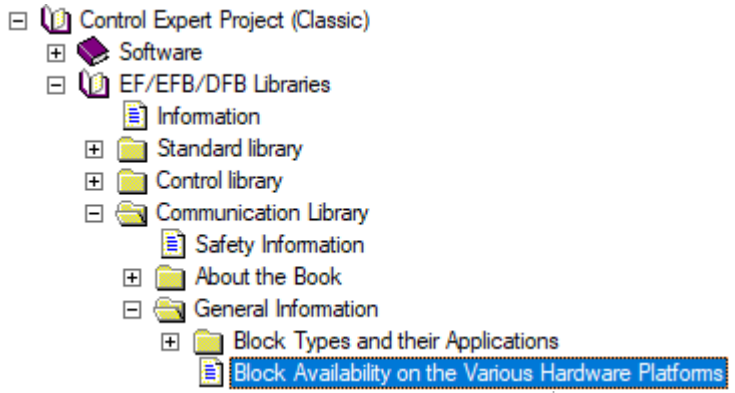
Some product names used in this manual are used for identification purposes only and may be trademarks of their respective companies.

Introduction – Block availability

> EcoStruxure Control Expert includes multiple explicit messaging blocks for Modbus -

- > Read_Var and Write_Var
  - \+ Easiest blocks to use for reading from 0x, 1x, 3x, 4x and writing to 0x, 4x registers
- > Mbp_Mstr
  - \+ Provides simplified functions for reading from and writing to 4x registers
  - \+ Also provides a function for sending any type of Modbus request
  - \+ Also provides a function for sending an Ethernet/IP message
  - \- Needs located variables for control and data registers
  - \- No inherent timeout functionality, has to be managed in the application
- > Data_Exch
  - \+ Allows sending any type of Modbus request
  - \+ Allows sending Ethernet/IP messages

Platform	Read_Var Write_Var	Mbp_Mstr	Data_Exch
Premium	Y	N	Y
Quantum	N	Y	N
M340	Y	Y	Y
M580	Y	N	Y



> For more info on block availability, refer to Control Expert's help.

Reminder - Explicit vs Implicit messaging

Explicit Messaging	Implicit Messaging
Implemented by blocks programmed in sections.	Implemented by IOScanning configured via DTMs.
Multiple blocks with the same target device use the same TCP connection.	Each IOScanning line for a target device needs a separate TCP connection.
CPU can use the ethernet bus or the Xbus (via another NOC module) to reach the target device.	Each communication module needs ethernet access to the target device.
Max simultaneously active blocks and TCP connections depends on the CPU model, cannot be expanded.	Each communication module (CPU, NOC) has IOScanning limits. Can be expanded by adding NOC modules (max 4, dependent on CPU model).
For Modbus, can use different function codes depending on the block.	For Modbus, uses FC23 if the target device supports it, else falls back to FC3 (Read 4x) and FC16 (Write 4x).
For Ethernet/IP, can use Connected or Unconnected communication modes.	For Ethernet/IP, always uses Class-1 I/O connection mode.
Can be configured/reconfigured online.	Workaround (HSBY_BUILD_OFFLINE block) available to make online changes only for M580 hot standby systems. For standalone and M340 systems, needs a full download.
Polling occurs in sync with the application task (e.g. MAST).	Polling cycle is independent of the application task cycle.
Suitable for architectures with multiple serial devices behind gateways.	Suitable for high-speed polling in networks with native ethernet devices.

Introduction

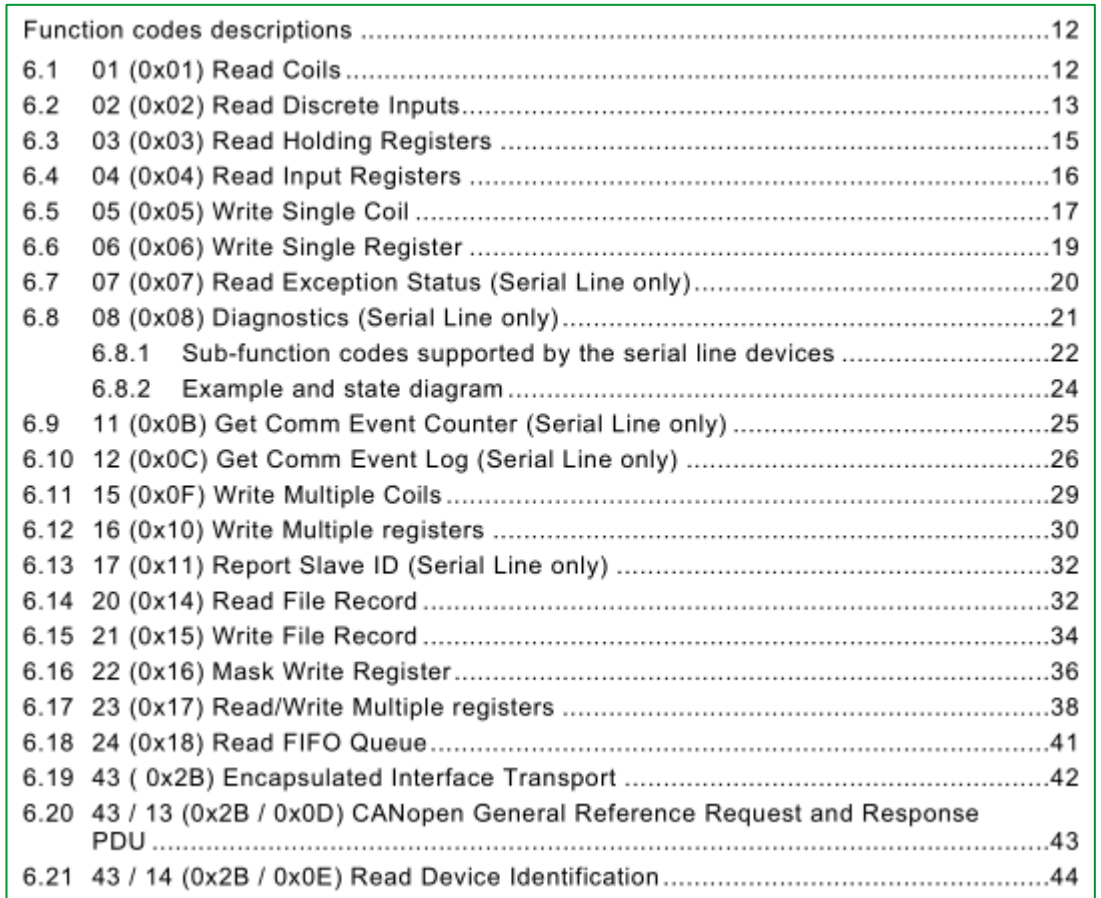
> Objective of this document is to show how to use Data_Exch to send any type of standard Modbus request. UMAS requests are out of scope.

> Steps are provided (with sample DFBs) to use Data_Exch for these function codes –
  > FC22: Mask Write Register
  > FC23: Read/Write Multiple Registers

> Similar steps can be followed for other types of Modbus requests. For implementation support, please contact your local Schneider Electric representative or technical support.

Modbus Protocol Reference

> To get implementation detail on the standard Modbus function codes, this document can be referred to –
  > https://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf



Function codes descriptions 12
6.1 01 (0x01) Read Coils 12
6.2 02 (0x02) Read Discrete Inputs 13
6.3 03 (0x03) Read Holding Registers 15
6.4 04 (0x04) Read Input Registers 16
6.5 05 (0x05) Write Single Coil 17
6.6 06 (0x06) Write Single Register 19
6.7 07 (0x07) Read Exception Status (Serial Line only) 20
6.8 08 (0x08) Diagnostics (Serial Line only) 21
6.8.1 Sub-function codes supported by the serial line devices 22
6.8.2 Example and state diagram 24
6.9 11 (0x0B) Get Comm Event Counter (Serial Line only) 25
6.10 12 (0x0C) Get Comm Event Log (Serial Line only) 26
6.11 15 (0x0F) Write Multiple Coils 29
6.12 16 (0x10) Write Multiple registers 30
6.13 17 (0x11) Report Slave ID (Serial Line only) 32
6.14 20 (0x14) Read File Record 32
6.15 21 (0x15) Write File Record 34
6.16 22 (0x16) Mask Write Register 36
6.17 23 (0x17) Read/Write Multiple registers 38
6.18 24 (0x18) Read FIFO Queue 41
6.19 43 (0x2B) Encapsulated Interface Transport 42
6.20 43 / 13 (0x2B / 0x0D) CANopen General Reference Request and Response PDU 43
6.21 43 / 14 (0x2B / 0x0E) Read Device Identification 44

Data_Exch – A primer

TYP :Transmission mode
Always to be set to 1, meaning "*transmission followed by await reception*"

EMIS: Request to be sent
Array of Integers, whose length and contents depend on the protocol and the function code.
Example, for Modbus, to write value 255 to %MW100-
From the Modbus manual on previous slide –

Function Code = 6, Write single Register		
Function Code	1 Byte	0x06 (6)
Register Address	2 Bytes	0x0064 (100)
Register Value	2 Bytes	0x00ff (255)

Note: Data is always transmitted as a stream of bytes. In Modbus, Big-Endian order is followed, hence for a 16-bit value, high byte is sent first. With Modicon PACs, Little-Endian order is followed, hence the low byte is sent first.

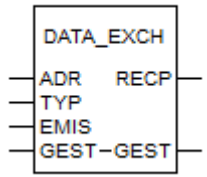
Byte order for Modbus request	
Byte #	Purpose
1	Function Code
2	Register Address MSB
3	Register Address LSB
4	Register Value MSB
5	Register Value LSB

Byte order for Modbus request in Modicon PACs			
Array Element	Byte	Purpose	Value
Emis[0]	MSB	Register Address MSB	0x00
	LSB	Function Code	0x06
Emis[1]	MSB	Register Value MSB	0x00
	LSB	Register Address LSB	0x64
Emis[2]	LSB	Register Value LSB	0xff

ADR: Address of Target device
Output of -
- ADDR (Premium)
- ADDM (M340/M580)
- ADDMX (x80 drop with M580)

- instructions can be linked here.

Example for Modbus: ADDM('r.m.c{IP Address}TCP.MBS')
Example for Ethernet/IP: ADDM('r.m.c{IP Address}UNC.CIP')



RECP: Received data

GEST: Management Table

Array Element	Byte	Purpose
Gest[0]	MSB	Exchange number
	LSB	Bit0: Activity flag Bit1: Cancel request
Gest[1]	MSB	Operation report
	LSB	Communication report
Gest[2]		Timeout, 100ms base
Gest[3]		Length of data to transmit / Length of data received

FC22: Mask Write Register

- This function is used to set or reset selected bits of a 4x register. The other bits are kept unchanged by the Modbus device.

- If this function is not used, and bit set/reset has to be performed, then the following transactions have to be done –
  - The Modbus client reads the register from the server.
  - The client modifies the bits in the read value.
  - The client writes the modified value back to the register. There is a possibility here that the Modbus server may have modified other bits which will be overwritten by the Modbus client.

- For using this function, the following parameters are sent by the Modbus client –
  - The register address (offset) for which bits are to be modified
  - An ‘And Mask’
  - An ‘Or Mask’

FC22: Mask Write Register

> This formula applied by the Modbus server is –
  > Result = (Current Contents AND And_Mask) OR (Or_Mask AND (NOT And_Mask))

> If the Or_Mask value is zero, the result is simply the logical ANDing of the current contents and And_Mask.

> If the And_Mask value is zero, the result is equal to the Or_Mask value.

> Example –

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	Current Value	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	=16#5555
To set bits 7,9 and reset bits 12,14	And_Mask	1	0	1	0	1	1	0	1	0	1	1	1	1	1	1	1	=16#AD7F
	Or_Mask	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	=16#0280
	Result	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0	1	=16#07D5

  > To set a bit: And_Mask_Bit = 0, Or_Mask_Bit = 1
  > To reset a bit: And_Mask_Bit = 0, Or_Mask_Bit = 0
  > To leave a bit unchanged: And_Mask_Bit = 1, Or_Mask_Bit = 0

FC22: Mask Write Register

Request

Function code	1 Byte	**0x16**
Reference Address	2 Bytes	0x0000 to 0xFFFF
And_Mask	2 Bytes	0x0000 to 0xFFFF
Or_Mask	2 Bytes	0x0000 to 0xFFFF

> On successful execution, the server sends a copy of the request back -

Response

Function code	1 Byte	**0x16**
Reference Address	2 Bytes	0x0000 to 0xFFFF
And_Mask	2 Bytes	0x0000 to 0xFFFF
Or_Mask	2 Bytes	0x0000 to 0xFFFF

> Or an exception code, if not successful -

Error

Error code	1 Byte	**0x96**
Exception code	1 Byte	01 or 02 or 03 or 04

FC22: Mask Write Register

> DFB data definition -

Name	Type	no.
DATA_EXCH_FC22	<DFB>	
<inputs>		
START	BOOL	1
ADR	ANY_ARRAY_INT	2
OFFSET	INT	4
AND_MASK	INT	5
OR_MASK	INT	6
TIMEOUT	TIME	8
<outputs>		
ACTIVE	BOOL	1
DONE	BOOL	2
ERROR	BOOL	3
COMM_ERROR_CODE	INT	4
MB_EXCEPTION_CODE	INT	5
<inputs/outputs>		
<public>		
<private>		
EMIS	ARRAY[0..3] OF INT	
GEST	ARRAY[0..3] OF INT	
RECP	ARRAY[0..3] OF INT	
R_TRIG_0	R_TRIG	
F_TRIG_0	F_TRIG	
OFFSET_LSB	INT	
OFFSET_MSB	INT	
AND_MASK_LSB	INT	
AND_MASK_MSB	INT	
OR_MASK_LSB	INT	
OR_MASK_MSB	INT	
TIMEOUT_INTE	INT	

FC22: Mask Write Register

> DFB section contents -

```
ACTIVE := GEST[0].0;
R_TRIG_0 (CLK := START);
F_TRIG_0 (CLK := ACTIVE);

IF AND_BOOL(R_TRIG_0.Q, NOT ACTIVE) THEN (* Trigger only when inactive *)

        (* Split data into bytes to pack in Modbus emission array *)
        OFFSET_LSB := AND_INT(OFFSET, 16#00FF);
        OFFSET_MSB := AND_INT(OFFSET, 16#FF00);
        AND_MASK_LSB := AND_INT(AND_MASK, 16#00FF);
        AND_MASK_MSB := AND_INT(AND_MASK, 16#FF00);
        OR_MASK_LSB := AND_INT(OR_MASK, 16#00FF);
        OR_MASK_MSB := AND_INT(OR_MASK, 16#FF00);

        EMIS[0] := OR_INT(OFFSET_MSB, 22);
        EMIS[1] := OR_INT(AND_MASK_MSB, OFFSET_LSB);
        EMIS[2] := OR_INT(OR_MASK_MSB, AND_MASK_LSB);
        EMIS[3] := OR_INT(0, OR_MASK_LSB);

        (* Timeout must be passed as hundredths of a Second *)
        TIMEOUT_INTE := TIME_TO_INT(DIVTIME_INT (IN1 := TIMEOUT, IN2 := 100));

        GEST[2] := TIMEOUT_INTE;
        GEST[3] := 7; (* Number of bytes containing data in the emission array.  EMIS[3] MSB is not used *)

        DONE := FALSE;
        ERROR := FALSE;

        DATA_EXCH (ADR := ADR, TYP := 1, EMIS := EMIS, GEST := GEST, RECP => RECP);
END_IF;
```

Refer request framing format on slide #10 and sample framing on slide #7

FC22: Mask Write Register

> DFB section contents -

```
IF F_TRIG_0.Q THEN
        COMM_ERROR_CODE := GEST[1];
        MB_EXCEPTION_CODE := 0;
        IF COMM_ERROR_CODE<>0 THEN
                ERROR := TRUE;
        END_IF;
        IF AND_INT(RECP[0],16#00FF) <> 22 THEN
                MB_EXCEPTION_CODE := SHRZ_INT (IN := RECP[0], N := 8);
                ERROR := TRUE;
        END_IF;
        DONE := TRUE;
END_IF;
```

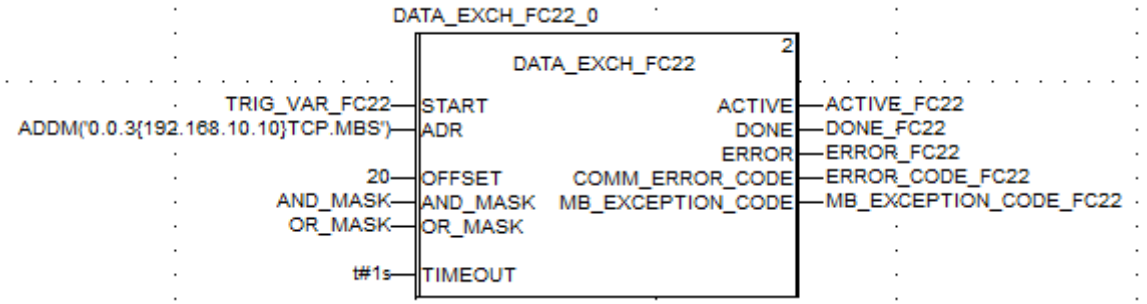
If an error code is returned here, it means the Modbus request could not be sent.

If exception is returned here, it means the Modbus device rejected the request.

01 = Function code not supported
02 = Bad address (offset)
03 = Bad And_Mask or Or_Mask
04 = Register write operation could not be processed

FC22: Mask Write Register

> Sample usage -



Before execution

Client

Name	Value
DATA_EXCH_FC22_0.EMIS	
DATA_EXCH_FC22_0.EMIS[0]	16#0000
DATA_EXCH_FC22_0.EMIS[1]	16#0000
DATA_EXCH_FC22_0.EMIS[2]	16#0000
DATA_EXCH_FC22_0.EMIS[3]	16#0000
DATA_EXCH_FC22_0.RECP	
DATA_EXCH_FC22_0.RECP[0]	16#0000
DATA_EXCH_FC22_0.RECP[1]	16#0000
DATA_EXCH_FC22_0.RECP[2]	16#0000
DATA_EXCH_FC22_0.RECP[3]	16#0000
TRIG_VAR_FC22	0
AND_MASK	16#AD7F
OR_MASK	16#0280
ACTIVE_FC22	0
DONE_FC22	0
ERROR_FC22	0
ERROR_CODE_FC22	0

Server

Name	Value	Type
%MW20	16#5555	INT

After execution

Client

Name	Value
DATA_EXCH_FC22_0.EMIS	
DATA_EXCH_FC22_0.EMIS[0]	16#0016
DATA_EXCH_FC22_0.EMIS[1]	16#AD14
DATA_EXCH_FC22_0.EMIS[2]	16#027F
DATA_EXCH_FC22_0.EMIS[3]	16#0080
DATA_EXCH_FC22_0.RECP	
DATA_EXCH_FC22_0.RECP[0]	16#0016
DATA_EXCH_FC22_0.RECP[1]	16#AD14
DATA_EXCH_FC22_0.RECP[2]	16#027F
DATA_EXCH_FC22_0.RECP[3]	16#0080
TRIG_VAR_FC22	1
AND_MASK	16#AD7F
OR_MASK	16#0280
ACTIVE_FC22	0
DONE_FC22	1
ERROR_FC22	0
ERROR_CODE_FC22	0

Server

Name	Value	Type
%MW20	16#07D5	INT

FC23: Read/Write Multiple Registers

> This function is used to read and write multiple 4x registers in a single message.
  > Max 125 registers can be read
  > Max 121 registers can be written

> The write operation is performed before the read, in the server.

> For using this function, the following parameters are sent by the Modbus client –
  > Starting address and the quantity for the read
  > Starting address for the write and the data to be written

FC23: Read/Write Multiple Registers

Request

Function code	1 Byte	**0x17**
Read Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity to Read	2 Bytes	0x0001 to 0x007D
Write Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity to Write	2 Bytes	0x0001 to 0X0079
Write Byte Count	1 Byte	2 x **N***
Write Registers Value	**N***x 2 Bytes	

***N** = Quantity to Write

Response

Function code	1 Byte	**0x17**
Byte Count	1 Byte	2 x **N'***
Read Registers value	**N'*** x 2 Bytes	

***N'** = Quantity to Read

FC23: Read/Write Multiple Registers

> DFB data definition -

Name	Type	no.
DATA_EXCH_FC23	<DFB>	
<inputs>		
START	BOOL	1
ADR	ANY_ARRAY_INT	2
RD_OFFSET	INT	4
RD_QTY	INT	5
RD_SWAP_BYTES	BOOL	6
WR_OFFSET	INT	8
WR_DATA	ANY_ARRAY_INT	9
WR_SWAP_BYTES	BOOL	10
TIMEOUT	TIME	12
<outputs>		
ACTIVE	BOOL	1
DONE	BOOL	2
ERROR	BOOL	3
COMM_ERROR_CODE	INT	4
MB_EXCEPTION_CODE	INT	5
ERROR_VAR_LENGTH	BOOL	6
RD_DATA	ANY_ARRAY_INT	9
<inputs/outputs>		
<public>		
<private>		
GEST	ARRAY[0..3] OF INT	
R_TRIG_0	R_TRIG	
F_TRIG_0	F_TRIG	
RD_OFFSET_LSB	INT	
RD_OFFSET_MSB	INT	
RD_QTY_LSB	INT	
RD_QTY_MSB	INT	
WR_OFFSET_LSB	INT	
WR_OFFSET_MSB	INT	
EMIS	ARRAY[0..125] OF INT	
TIMEOUT_INTE	INT	
LEN_WR_DATA	INT	
i	INT	
RECP	ARRAY[0..125] OF INT	
BYTES_REC	INT	
LEN_RD_DATA	INT	
LEN_WR_DATA_LSB	INT	
LEN_WR_DATA_MSB	INT	

FC23: Read/Write Multiple Registers

> DFB section contents -

```
ACTIVE := GEST[0].0;
R_TRIG_0 (CLK := START);
F_TRIG_0 (CLK := ACTIVE);

IF AND_BOOL(R_TRIG_0.Q, NOT ACTIVE) THEN (* Trigger only when inactive *)

        (* Split data into bytes to pack in Modbus emission array *)
        RD_OFFSET_LSB := AND_INT(RD_OFFSET, 16#00FF);
        RD_OFFSET_MSB := AND_INT(RD_OFFSET, 16#FF00);
        RD_QTY_LSB := AND_INT(RD_QTY, 16#00FF);
        RD_QTY_MSB := AND_INT(RD_QTY, 16#FF00);
        WR_OFFSET_LSB := AND_INT(WR_OFFSET, 16#00FF);
        WR_OFFSET_MSB := AND_INT(WR_OFFSET, 16#FF00);
        LEN_WR_DATA := LENGTH_ARINT(WR_DATA);
        LEN_WR_DATA_LSB := AND_INT(LEN_WR_DATA, 16#00FF);
        LEN_WR_DATA_MSB := AND_INT(LEN_WR_DATA, 16#FF00);

        EMIS[0] := OR_INT(RD_OFFSET_MSB, 23);
        EMIS[1] := OR_INT(RD_QTY_MSB, RD_OFFSET_LSB);
        EMIS[2] := OR_INT(WR_OFFSET_MSB, RD_QTY_LSB);
        EMIS[3] := OR_INT(LEN_WR_DATA_MSB, WR_OFFSET_LSB);
        EMIS[4] := OR_INT(SHL_INT(IN :=LEN_WR_DATA*2 , N := 8), LEN_WR_DATA_LSB);
        FOR i := 0 TO LEN_WR_DATA-1 BY 1 DO
                IF WR_SWAP_BYTES THEN
                        EMIS[5+i] := ROL(WR_DATA[i],8);
                ELSE
                        EMIS[5+i] := WR_DATA[i];
                END_IF;
        END_FOR;

        (* Timeout must be passed as hundredths of a Second *)
        TIMEOUT_INTE := TIME_TO_INT(DIVTIME_INT (IN1 := TIMEOUT, IN2 := 100));

        GEST[2] := TIMEOUT_INTE;
        GEST[3] := 10 + LEN_WR_DATA*2;

        DONE := FALSE;
        ERROR := FALSE;

        DATA_EXCH (     ADR := ADR, TYP := 1, EMIS := EMIS, GEST := GEST, RECP => RECP);

END_IF;
```

Refer request framing format on slide #16 and sample framing on slide #7

Byte swap may be required depending on Little-Endian or Big-Endian ordering in the device

FC23: Read/Write Multiple Registers

> DFB section contents -

```
IF F_TRIG_0.Q THEN
        COMM_ERROR_CODE := GEST[1];
        MB_EXCEPTION_CODE := 0;
        IF COMM_ERROR_CODE=0 THEN
                IF AND_INT(RECP[0],16#00FF)=23 THEN
                        BYTES_REC := SHRZ_INT (IN := RECP[0], N := 8);
                        LEN_RD_DATA := LENGTH_ARINT (IN := RD_DATA);
                        IF LEN_RD_DATA >= BYTES_REC/2 THEN
                                FOR i := 0 TO BYTES_REC/2 - 1 BY 1 DO
                                        IF RD_SWAP_BYTES THEN
                                                RD_DATA[i] := ROL(RECP[1+i],8);
                                        ELSE
                                                RD_DATA[i] := RECP[1+i];
                                        END_IF;
                                END_FOR;
                        ELSE
                                ERROR := TRUE;
                        END_IF;
                ELSE
                        MB_EXCEPTION_CODE := SHRZ_INT (IN := RECP[0], N := 8);
                        ERROR := TRUE;
                END_IF;
        ELSE
                ERROR := TRUE;
        END_IF;
        DONE := TRUE;
END_IF;
```

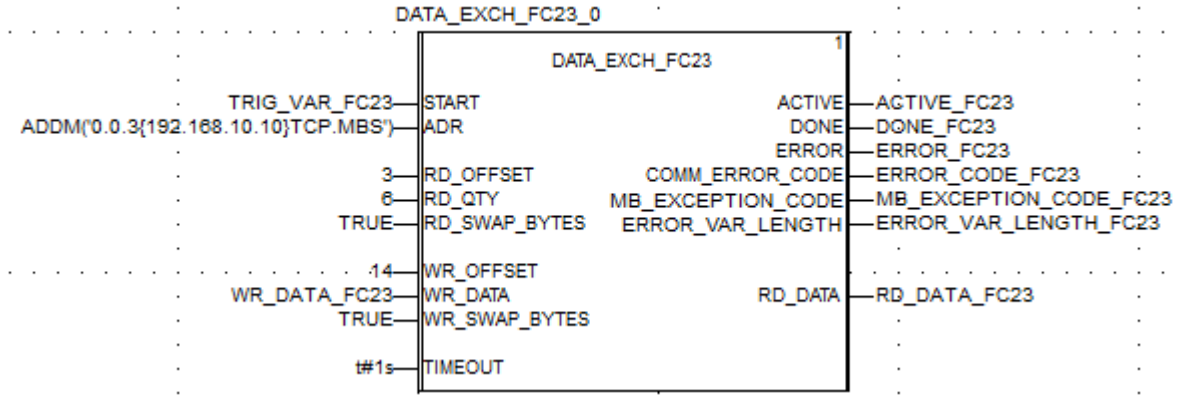
If an error code is returned here, it means the Modbus request could not be sent.

If exception is returned here, it means the Modbus device rejected the request.

01 = Function code not supported
02 = Read or write address space exceeded
03 = (Read length > 125) or (Write length >121)
04 = Register read/write operation could not be processed

FC23: Read/Write Multiple Registers

> Sample usage -



Before execution

Client

Name	Value	Type
TRIG_VAR_FC23	0	BOOL
WR_DATA_FC23		ARRAY[0..2] OF INT
WR_DATA_FC23[0]	101	INT
WR_DATA_FC23[1]	201	INT
WR_DATA_FC23[2]	301	INT
DONE_FC23	0	BOOL
ERROR_FC23	0	BOOL
RD_DATA_FC23		ARRAY[0..5] OF INT
RD_DATA_FC23[0]	0	INT
RD_DATA_FC23[1]	0	INT
RD_DATA_FC23[2]	0	INT
RD_DATA_FC23[3]	0	INT
RD_DATA_FC23[4]	0	INT
RD_DATA_FC23[5]	0	INT

Server

Name	Value	Type
%MW3:6		ARRAY[0..5] OF INT
%MW3[0]	1	INT
%MW3[1]	2	INT
%MW3[2]	3	INT
%MW3[3]	4	INT
%MW3[4]	5	INT
%MW3[5]	6	INT
%MW14:3		ARRAY[0..2] OF INT
%MW14[0]	0	INT
%MW14[1]	0	INT
%MW14[2]	0	INT

After execution

Client

Name	Value	Type
TRIG_VAR_FC23	1	BOOL
WR_DATA_FC23		ARRAY[0..2] OF INT
WR_DATA_FC23[0]	101	INT
WR_DATA_FC23[1]	201	INT
WR_DATA_FC23[2]	301	INT
DONE_FC23	1	BOOL
ERROR_FC23	0	BOOL
RD_DATA_FC23		ARRAY[0..5] OF INT
RD_DATA_FC23[0]	1	INT
RD_DATA_FC23[1]	2	INT
RD_DATA_FC23[2]	3	INT
RD_DATA_FC23[3]	4	INT
RD_DATA_FC23[4]	5	INT
RD_DATA_FC23[5]	6	INT

Server

Name	Value	Type
%MW3:6		ARRAY[0..5] OF INT
%MW3[0]	1	INT
%MW3[1]	2	INT
%MW3[2]	3	INT
%MW3[3]	4	INT
%MW3[4]	5	INT
%MW3[5]	6	INT
%MW14:3		ARRAY[0..2] OF INT
%MW14[0]	101	INT
%MW14[1]	201	INT
%MW14[2]	301	INT

Sample Application

> An FBD section with the sample DFBs can be downloaded from

https://schneider-electric.box.com/s/23fkotg66aqaulcd5jzg9ybmfqcok29x

> These are not official DFBs, hence users should fully test them before using in their projects to avoid unintended consequences.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Always test additional DFBs before deployment to the Production environment.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Life Is On Schneider Electric