SNT
Le mouvement bien pensé

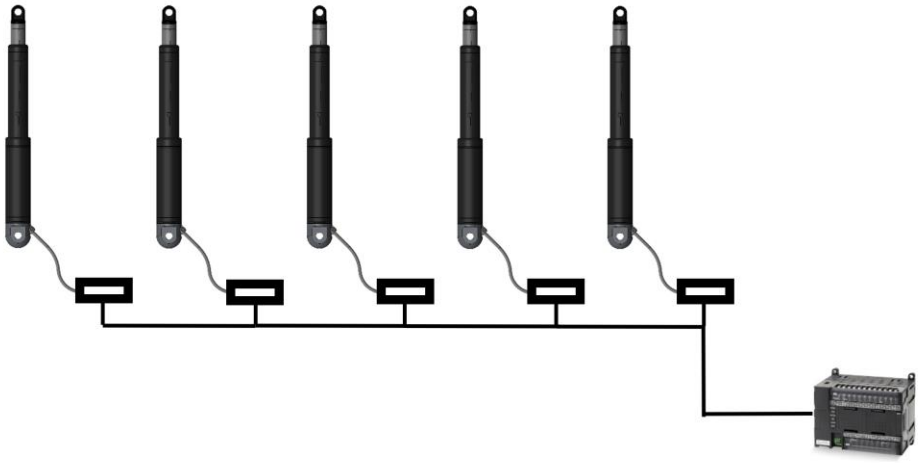
Icon
actionneur en ligne

Spécifications préliminaires
pour actionneurs Icon Modbus RTU

The RS485 MODBUS RTU option is a serial communication interface between the actuators and a control system. The MODBUS interface can directly communicate with a PLC with a MODBUS module or a PC through an external USB to RS485 interface box.
This document describes how to install, configure and use an actuator with embedded MODBUS RTU serial communication.
Basic serial bus communication knowledge is a prerequisite for using and understanding the below documentation.

Concept Modbus RTU System

Universally recognized and widely used, the MODBUS RTU fieldbus is still an essential, open communications standard, supported by a large number of products on the market today.
In the MODBUS network any MODBUS Master can be connected to one or several Concens icon Actuators with MODBUS interface. The topology is a serial bus-system with actuators including derivation cable connected to a trunk cable through passive TAPs. One master PLC or PC can be connected to the serial bus to control and supervise the actuator slaves. The actuators might potentially be mixed with other 3rd party MODBUS slaves.



At the physical protocol level, the RS-485 (TIA/EIA-485) two-wire interface is used, which supports half-duplex communication between the master and one or more slaves. Inter-communication between slaves is not possible and a slave will never transmit data without receiving a request from the master. The master (incl. terminator) can be connected at the end of the cable as shown in the figure above.
Alternatively, connection of the master anywhere in-between two of the slaves is acceptable as well.

Safety instructions

Be aware of the following symbols throughout the installation guide:

Recommendations

Failing to follow these instructions can result in the actuator suffering damage or being ruined.

Additional information

Usage tips or additional information that is important in connection with the use of the actuator.

Be aware that a lot of test and quality activities have been performed to ensure the functionality and safe use of the product. As with other electronic equipment the MODBUS option has a finite failure rate. To ensure that one failure does not lead to an unsafe state the microcontroller monitors critical components (1. failure surveillance). It is still possible to run the actuator but it is of utmost importance that the user reacts upon these events to maintain 1. failure safety (i.e. polling of relevant MODBUS registers to identify the reason for last stop)

Connection, cables, and plugs

The actuator data and power cables are integrated and pre-mounted on the Concens icon Actuators.

Modbus RTU Functional Description

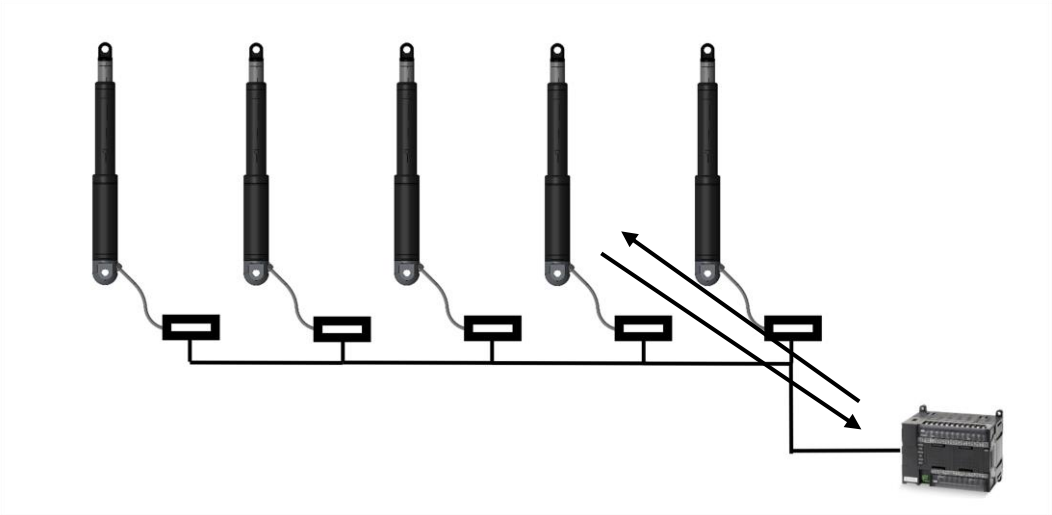
The Concens MODBUS protocol is implemented conforming to the “basic slave” implementation class. The following options have been implemented:

Parameter	Options	Default value	Remarks
Adressing	Configurable from 1 to 246	8 (= un-assigned)	Configured by CAS software
	-		
Baud rate	-	115.2 kBaud	
Parity	-	None	
Stop Bits	-	1	
Mode	RTU	-	
Electrical Interface	RS485 2W-cabling	-	
Connector Type	Yes	Molex 8 pol Mini Fit JR	

Unicast/broadcast

MODBUS is a single master system, which means that only one master can be connected at any single point in time. Two modes of communication are possible, Unicast and Broadcast.

Request/response (unicast)
The requests from the master are addressed to a given slave. The master then waits for the response from the slave which has been interrogated. In this mode the transaction consists of 2 messages: a request from the master and a response from the slave.

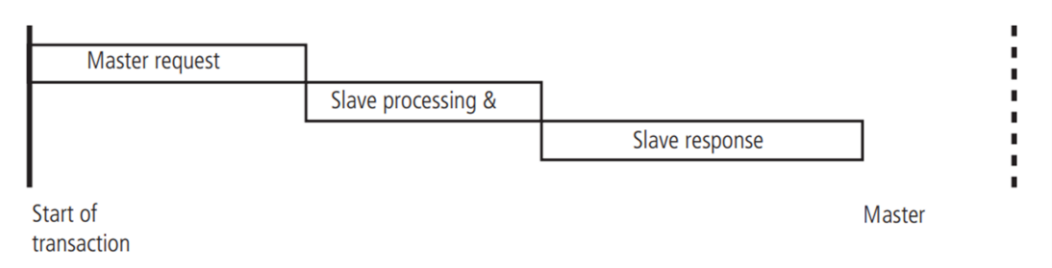


Broadcasting
No Broadcasting is possible.

Response time

The slave device will respond on each valid MODBUS request from the master within a time which is dependent on the setting of parameter 'MODBUS Response Delay' (Input Register XXXX). With a default parameter value (3 ms) the max. response time is 18 ms (3 + 15 ms) – when increasing the parameter value worst case is 115 ms (100 + 15 ms).

Modbus message timing:



The MODBUS response timeout setting of the master should be set to a value larger than the calculated max. response time.

Error checking

MODBUS RTU networks employ two methods of error checking:
1. Parity checking of each data character (no parity)
2. Frame checking within the message frame (Cyclical Redundancy Check)

Parity checking
A Concens MODBUS device can only be configured for no parity checking. This determines how the parity bit of the character's data frame is set.

Frame checking
RTU Mode message frames include an error checking method that is based on a Cyclical Redundancy Check (CRC). The error-checking field of a message frame contains a 16-bit value (two 8-bit bytes) that contains the result of a Cyclical Redundancy Check (CRC) calculation performed on the message contents.

Message format

Any MODBUS message consists of the basic fields shown below: Slave address (Addr), Function code (Function), up to 252 data bytes (Data) and a calculated 16 bit checksum (CRC).

Byte 0	Byte 1	Byte 2 .. N	Byte N+1, N+2
Addr	Function	Data	CRC

Messages start with a silent interval of at least 3.5 character times – at the actual communication baud rate. The first field transmitted is the device address. Following the last transmitted byte, a similar interval of at least 3.5 character times marks the end of the message. A new message can begin after this interval.

Address field

The address field Addr is one byte long. Valid slave addresses are 1 – 246. Value 0 and value 200 to 255 is reserved for special purposes. A master addresses a slave by placing the slave address in the Addr field of the message. When the slave responds it places its own address in the Addr field to let the master know which slave is responding.

Each slave device must have assigned a unique address (from 199) so that it can be addressed independently from other nodes.

Function field

The function field Function is one byte. Supported MODBUS functions are 3, 16. When a message is sent from the master to a slave, the function field code tells the slave what kind of action to perform.

When the slave responds to the master, it uses the function field to signal either a normal, error-free response or an exception response. For a normal response the slave simply echoes the original function code. For an exception response the slave returns a code that is equivalent to the original function code with the most significant bit set. In addition, the slave adds a unique code into the data field of the message telling the master what kind of error occurred.

Data field

The data field is of varying length. The data field of the message sent from the master to the slave contains additional information which the slave must use to take the action requested by the function field. This can include items like register addresses, quantity of register to handle, and the count of actual data bytes in the message.

CRC field

The CRC field is 2 bytes long. The CRC value is calculated by the transmitting device, which appends it to the end of the message. The receiving device recalculates a CRC during receipt or the message and compares the calculated value to the actual value received in the CRC field. In the case of a difference an exception response is returned.

Register-parameter mapping

All data addresses in MODBUS messages to Concens actuators are referenced to zero. The input register known as e.g. 'Input Register 1XXX' in a programmable controller is addressed as register 1XXX in the Addr field of a MODBUS message. The function code of the message already specifies an 'Input Register' operation and therefore the 'xxxx' reference is implicit. Page XX of XX In the same way the holding register known as e.g. 'Holding register 1XXX' is addressed as register 1XXX in the Addr field of the message, together with a relevant Holding Register function code.

All MODBUS registers are default mapped to a specific actuator parameter.

Data formats

Application information communicated between MODBUS master and slave is organized as one or more 16-bit Registers. Different datatypes are mapped into theses addressable registers. The type of any parameter value embedded into the MODBUS message has to be recognized according to the register/parameter tables in Appendix A. Concens MODBUS devices support the following datatypes (illustrated by single register write-message examples):

Short integer register (U8)

Status bytes and small integer values are stored in MODBUS registers where only half of the register is utilised. Integer values from 0 to 255 are stored in the least significant byte of the register.

Byte 0	Byte 1	Byte 2 .. 5				Byte 6, 7	
Addr	Function	Register address		Register data		CRC	
		MSB	LSB	N/A	U8		

Unsigned integer register (U16)

Integer values from 0 to 65,535 are stored in the normal 2-byte MODBUS register. The most significant byte of the value is sent first.

Byte 0	Byte 1	Byte 2 .. 5				Byte 6, 7	
Addr	Function	Register address		Register data		CRC	
		MSB	LSB	U16-MSB	U16-LSB		

Signed integer register (S16)

Integer values from -32,768 to 32,767 are stored in the normal 2-byte MODBUS register. The most significant byte of the value is sent first.

Byte 0	Byte 1	Byte 2 .. 5				Byte 6, 7	
Addr	Function	Register address		Register data		CRC	
		MSB	LSB	S16-MSB	S16-LSB		

Long integer registers (U32)

Some integer values used by the actuator are larger than 65,535, which is the largest number that can be stored in a single MODBUS register. In these cases, the unsigned value is stored in two consecutive integer registers enabling values up to 4,294,967,295. These “long integer” registers are potentially accessed using the MODBUS functions 3, 4, and 16 (see below). The most significant word is stored in the lower register (sent first), and the least significant word is stored in the higher register.

Byte 0	Byte 1	Byte 2 .. 7						Byte 8, 9	
Addr	Function	Register address		Register data				CRC	
		MSB	LSB	U32-MSB			U32-LSB		

Function codes

The Concens actuator supports a subset of the standard MODBUS RTU function codes to provide access to the internal actuator parameters and functions.

The Concens MODBUS protocol does not support the Diagnostic function (function code 08). As a more general approach, the MODBUS master – and the service tool – can read a large amount of service counter input registers.

Function code 3 – Read Holding Registers

Function code 3 is used to read one or more holding registers in the actuator, referenced in Appendix A. When the master accesses a register that is not supported by the slave it responds with an exception message. The register address in this context is without 'xxxx' identification. E.g. the 'Target Position' holding register is read via register address XXXX – in the PLC often referenced as XXXX. Broadcast is not supported. Request message:

Addr	Function	Register address		Register count		CRC	
Slave Address	Function Code (=3)	Starting Address High	Starting Address Low	No. of Registers High	No. of Regis-ters Low		

In the response message the requested values are delivered to the master. The data bytes are organized according to data type as explained above.

Response message:

Addr	Function		Data	CRC	
Slave Address	Function Code (=3)	Byte Count	Data byte 1 .. N		

Function code 16 – Write Multiple Holding Registers

Function code 16 is used to write one or more holding registers in the actuator, referenced in Appendix A. When the master accesses a register that is not supported by the slave it responds with an exception message.

The register address in this context is without 'xxxx' identification, e.g. meaning address 0 equals register xxxx.

Multiple registers are written as an entity with commands executed as the final stage. That means, if e.g. Holding Register xxx1, xxx2 and xxx3 (Target Position, Command-Remote and Max Speed) are written in one single command then the actuator will start running towards the new position with the new speed.

Broadcast is not supported.

Request message:

Addr	Function	Register address		Register count			Register Data	CRC	
Slave Address	Function Code (=3)	Starting Address High	Starting Address Low	No. of Registers High	No. of Registers Low	Byte Count	New Value 1 .. N		

The normal response message returns the slave address, function code, starting address and the quantity of registers written.

Response message:

Addr	Function	Register address		Register count		CRC	
Slave Address	Function Code (=16)	Starting Address High	Starting Address Low	Register Count High	Register Count Low		

Exceptions

When a MODBUS master transmits a request to a slave the expected behaviour is that the slave responds with a normal response. But several error scenarios are possible:

• The slave does not receive the request due to a communication error. No response is returned; the master will process a time-out and eventually repeat the request.

• The slave receives the request but detects a parity or CRC communication error. No response is returned; the master will process a time-out and eventually repeat the request.

Parallel position control

Definition of devices on the MODBUS in parallel setup

- • Master, the one on the MODBUS who initiates a parallel positioning (e.g. CAS).
- • Primary actuator, the one who controls the other actuators
- • Slave actuators, the ones who is following the primary actuator

A parallel positioning can both be initiated from the MODBUS via e.g., CAS or by the analogue interface on the primary actuator.
Following is required to run in parallel mode with the actuators.

1. Up to 8 actuators can run in parallel
2. They all have to be on the same MODBUS
3. No other actuators are allowed to join the MODBUS, i.e. all the actuators (max 8) on the physical MODBUS has to join the parallel positioning
4. One actuator is pointed out to be the primary, i.e. the one with MODBUS address 200
5. Up to 7 slaves can be configured to follow the primary actuator. The slaves must have an address from 201-207
6. When a parallel run is activated via the analogue interface on the primary actuator (i.e. the DIR buttons or analog input value changes) we call it *"local initiated positioning"* from now, the MODBUS master (e.g. CAS) MUST also leave the bus. I.e. "Concens Actuator Studio" is not allowed to "poll" anything.

In parallel position control where more than one actuator has to follows, the *position control* de-scribed in section 3.2 is used. To detect if an actuator is blocked or has problems following the position profile the status signal Err-slip is used.
I.e. each actuator has it owns closed position loop and if it does not follow the position profile (if the Err-slip is activated) the actuator will report an error (when polled by the primary actuator) and all actuator are requested to stopped by the primary actuator.
When a parallel *local initiated positioning* is activated on the primary actuator, it is expected the MODBUS master is quite again on the bus. I.e. before the *local initiated positioning* is activated the MODBUS master has been stopped sending requests (polling etc.) on the bus.

The sequence diagram below illustrates how a parallel *local initiated positioning* works.

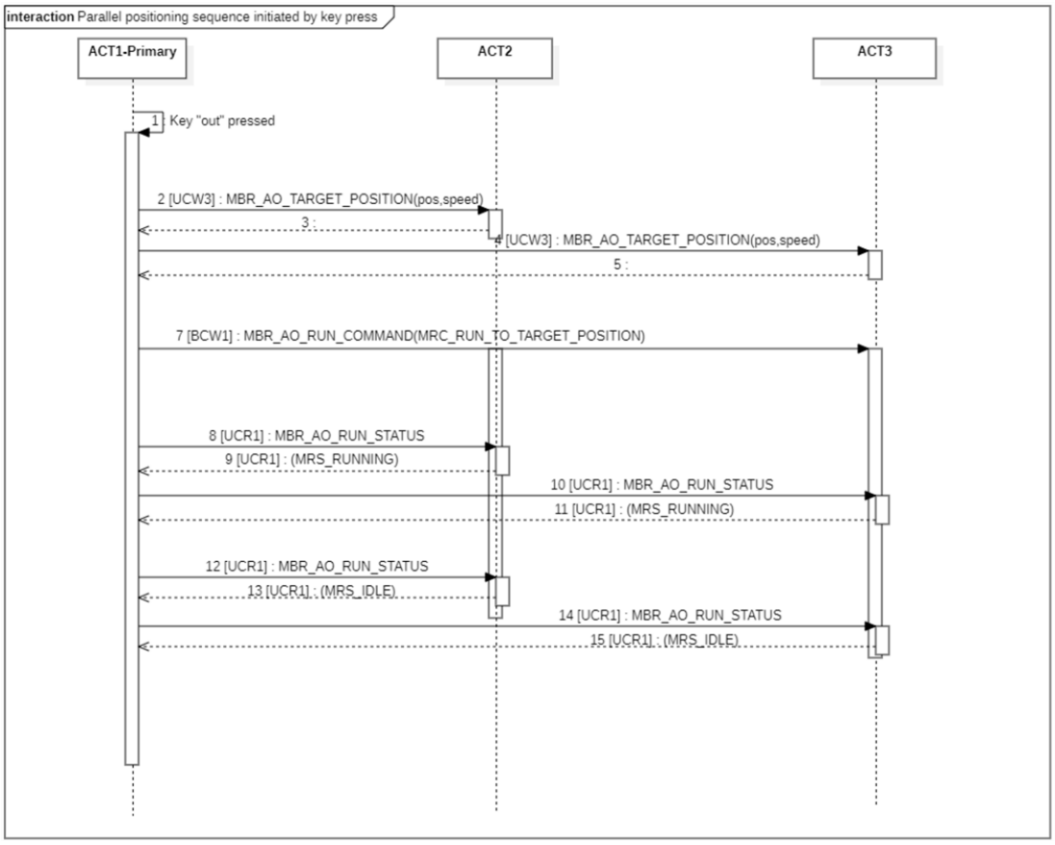


Figure Parallel run initiated by local key press

The same protocol is used as for the MODBUS master-initiated case, but here the MODBUS master MUST be the “heartbeat” creator, by polling the primary actuator. Below the sequence diagram illustrates what happens when the direction button on the primary actuator is released before the position is reached.

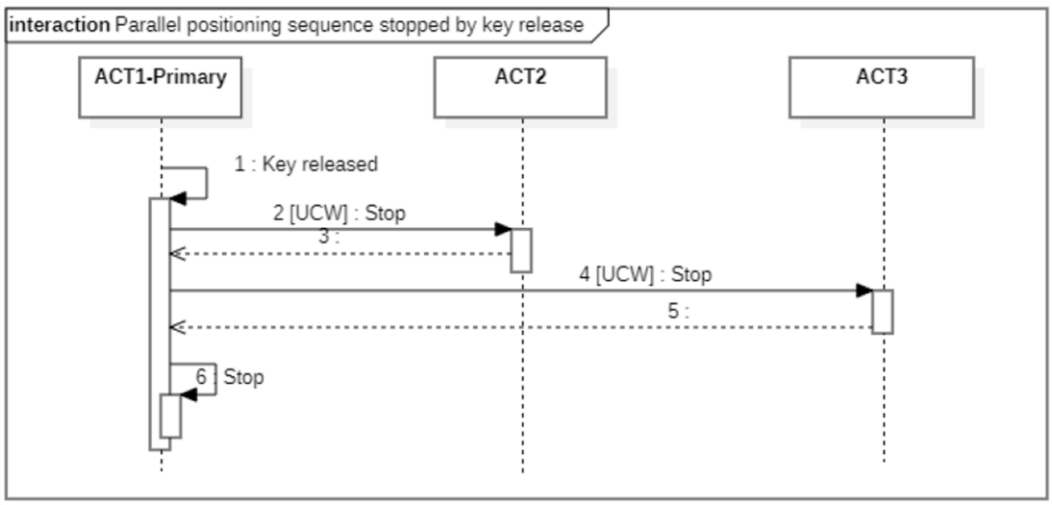


Figure Parallel stop initiated by local key release

If an actuator had problem following the velocity ramp, it will report an “Slip” error to the primary actuator when he polls for the status. The primary actuator will request all actuators on the bus to stop.
See the sequence diagram example below

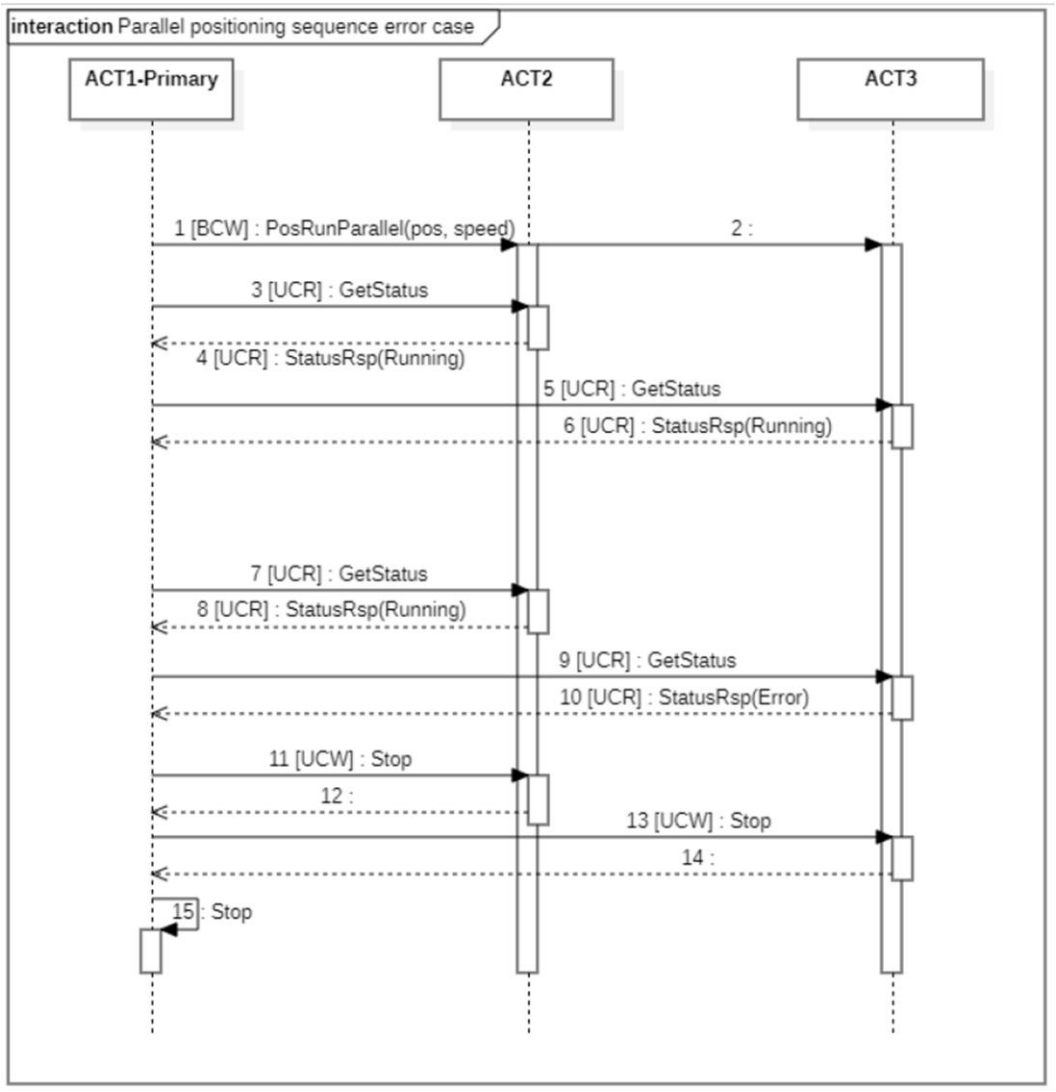


Figure Parallel run where an error occurs

Below is a sequence diagram of how the MODBUS master initiates the parallel run.

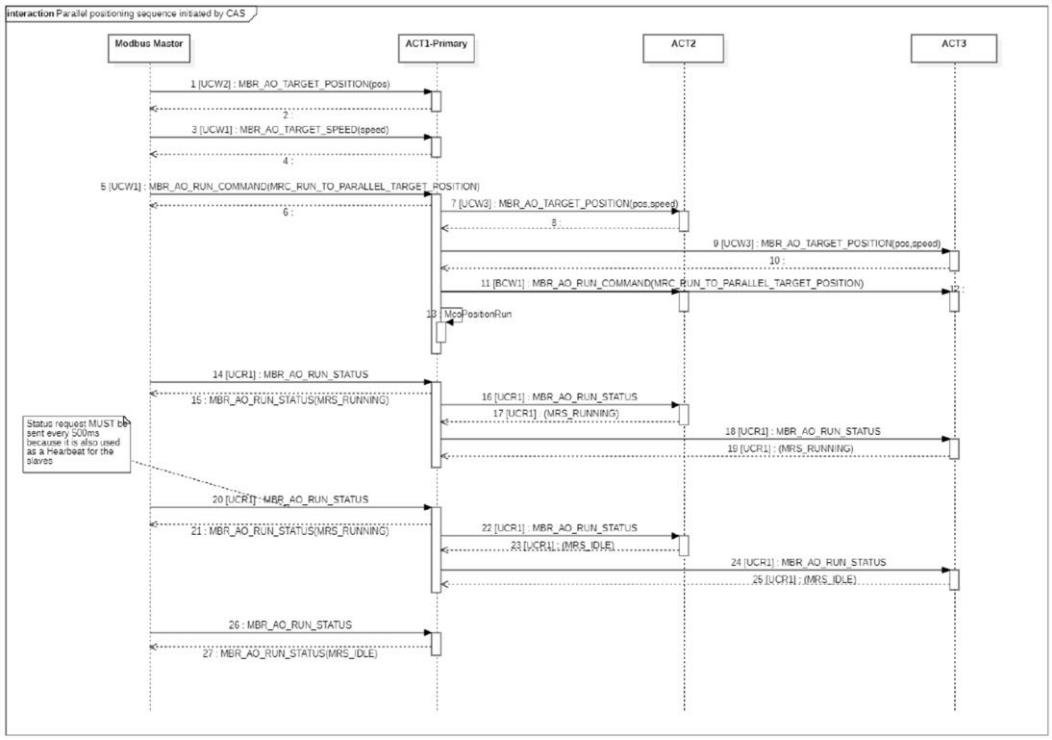
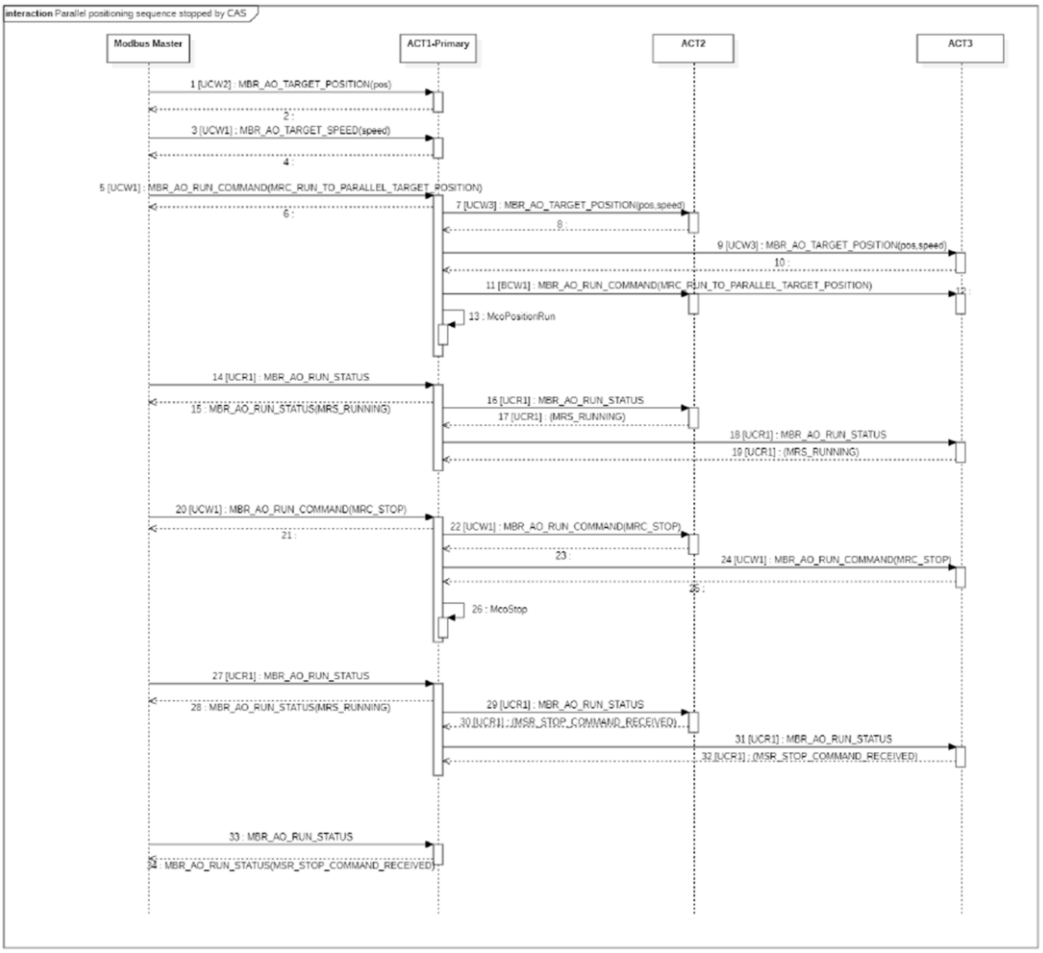


Figure Parallel run initiated by MODBUS Master

The MODBUS master MUST poll the primary actuator for status every 500ms to generate a “Hear-beat” event, which the primary actuator uses to poll all the slaves and return one single status response to the master. I.e. if the MODBUS master stops polling/generating the heartbeat all actuators will stop within 1s.

This also means that a 1s. delay before other actuators will react on a fault on another actuator. The delay will result in a mechanical misalignment if a fault occurs.

Below is an example of the MODBUS master which requests a parallel stop.



Modbus General

Only unicast messages generate a response from the device in question, Broadcast messages does not!

All the registers are read- and writable, but some of the register ignore write request (identity, run status etc.).

All the registers are unsigned 16-bit (uint16_t), together (xx + xx_HIGH) 32-bit registers are supported:

uint16_t u16 = reg[n];

int16_t i16 = (int16_t)reg[n];

uint32_t u32 = *(uint32_t*)®[n]; // and reg[n+1]

int32_t i32 = *(int32_t*)®[n]; // and reg[n+1]

When more registers are written in one go (function 16), and if one or more exceptions occurs, only the last execution status is responded back (MB_EX_ILLEGAL_DATA_ADDRESS, MB_EX_SLAVE_BUSY, or MB_EX_SLAVE_DEVICE_FAILURE).

Reading “write only” (WO) register return 0, no exception is raised! Writing “read only” (RO) register is just ignored; no exception is raised!

All registers are resat when the Broadcast messages “Reset configuration”. There values are zero (0) unless another default value are mentioned in the following register lists.

Modbus Registers

These unicast registers are intended used during the

- end-customers normal operation
- service setup

Register	Addr.	Data Interpretation	
Uptime **MBR_AN_UPTIME**	1000	uint32_t msec since last boot	FC3: Is running for how long time.
Product, type and firmware version **MBR_AN_PRODUCT**	1002	uint32_t **RO** The below areas are OR'ed together: Product: • • CON35 0x**10**000000 • • CON50 0x20000000 • • CON60 0x30000000 Nominal Supply Voltage • • 12 Volt 0x**01**000000 • • 24 Volt 0x02000000 • • 36 Volt 0x03000000 HW Revision • • Ver 1: 0x00**1**00000 FW Version • • Ver 2.1 0x000**21**000 max (15.15) Speed Control • • Base: 0x0000**1**00 • • Enh.: 0x00000200 Some parameters are set by: MBR_AC_CONFIGURATION	FC3: Information of setup.
Current **MBR_AN_CURRENT**	1004	Idle current + Motor Current Value in mAmp	FC3: Measurement
Run Status **MBR_AN_EXECUTE_STATUS**	1005	MbrExecuteStatus_t 1 MRS_IDLE 2 MRS_RUNNING_OUT 3 MRS_RUNNING_IN 4 MRS_WAITING_FOR_TIME 5 MRS_INITIALISING_HOME 6 MRS_STOPPING 7 MRS_SCANNING 8 MRS_POS_LEARNING 9 MRS_SPEED_LEARNING 10 MRS_RAMP_LEARNING 11 MRS_PID_TUNING 12 MRS_FACTORY_TEST_RUNNING	FC3: Running status
Actual Position **MBR_AN_ACTUAL_POSITION**	1006	int32_t Encoder Counts	FC3: Position counts
Distance from Target **MBR_AN_DISTANCE_FROM_TARGET**	1008	int32_t Encoder Counts	FC3: Counts from end position.
Reason for last stop **MBR_AN_REASON_FOR_STOP**	1010	MbrStopReason_t 1 MSR_TARGET_POSITION_REACHED 2 MSR_STOP_COMMAND_RECEIVED 3 MSR_HW_UNDERVOLTAGE 4 MSR_HW_STOP 5 MSR_ILLIGAL_POSITION 6 MSR_ILLIGAL_VELOCITY 7 MSR_ILLIGAL_ACCELERATION 8 MSR_CURRENT_CUTOFF_OUT 9 MSR_CURRENT_CUTOFF_IN 10 MSR_UNDERVOLTAGE_DETECTED_ON_V_IN 11 MSR_OVERVOLTAGE_DETECTED_ON_V_IN 12 MSR_ENCODER_ERROR 13 MSR_SLIP_ERROR 14 MSR_TEMPERATURE_EXCEEDED 15 MSR_HOME 16 MSR_BOOTED 17 MSR_IS_RUNNING 18 MSR_TIMEOUT 19 MSR_BUS_ERROR	FC3: Stop status- Why has it stopped the last time.

Next Time to Run **MBR_AN_NEXT_TIME_TO_RUN**	1011	Seconds delay before executing MRC_RUN_TO_TARGET_POSITION	FC3/FC16: Delay for start.
Reference position 1 **MBR_AN_REFERENCE_POSITION_1**	1012	int32_t Encoder Counts	FC3/FC16: Programable position 1
Reference position 2 **MBR_AN_REFERENCE_POSITION_2**	1014	int32_t Encoder Counts	FC3/FC16: Programable position 2
Reference position 3 **MBR_AN_REFERENCE_POSITION_3**	1016	int32_t Encoder Counts	FC3/FC16: Programable position 3
Reference position 4 **MBR_AN_REFERENCE_POSITION_4**	1018	int32_t Encoder Counts	FC3/FC16: Programable position 4
Target Position **MBR_AN_TARGET_POSITION**	1020	int32_t Encoder Counts +/- relative to home position	FC3/FC16: Target position
Max Speed (Run speed after ramping up) **MBR_AN_TARGET_SPEED**	1022	% of max possible speed	FC3/FC16: Speed in % normal 100%
Execute Command MBR_AN_EXECUTE_COMMAND	1023	**MbrExecuteCommand_t** **1 = MRC_STOP** **2 = MRC_RUN_TO_TARGET_POSITION**	FC16: Execution command.
		3 = MRC_RUN_TO_REFERENCE_POSITION_1 **4 = MRC_RUN_TO_REFERENCE_POSITION_2** **5 = MRC_RUN_TO_REFERENCE_POSITION_3** **6 = MRC_RUN_TO_REFERENCE_POSITION_4** **7 = MRC_RUN_AT_NEXT_TIME_TO_RUN** **8 = MRC_RUN_TO_PARALLEL_TARGET_POSITION,** **9 = MRC_STOP_PARALLEL** **10 = MRC_SCAN_SLAVES** **11 = MRC_POS_HOME** **12 = MRC_POS_LEARN** **13 = MRC_SPEED_LEARN** **14 = MRC_RAMP_LEARN** **15 = MRC_PID_TUNE** **16 = MRC_POSITION_SET** (set target position as new current position **17 = MRC_RUN_FACTORY_TEST** **18 = MRC_REBOOT // Also valid in broadcast** **19 = MRC_RESET_CONFIG //do**	
Speed during Learn and Home **MBR_AN_LEARN_SPEED**	1024	% of max possible speed	FC3/FC16: Speed when doing learning or homing in % normal 50%
Retracted Offset (max in position) **MBR_AN_HOME_OFFSET**	1025	uint32_t / Encoder Counts Home Position according to "MBR_AN_HOME_POS" Error if: (MBR_AN_EXTENDED_OFFSET - MBR_AN_RETRACTED_OFFSET) > MBR_AC_MAX_STROKE_LENGTH Default: 50	FC3/FC16: When overcurrent occur, it will drive back.
Ramp Down Before Target **MBR_AN_RAMP_DOWN_BEFORE_TARGET**	1027	Resolution in encoder steps Default 20 counts	
MBR_AN_SAFETY_ZONE_FORWARD	1028	The maximum position using the Forward Button Input Default: MBR_AC_MAX_STROKE_LENGTH	FC3/FC16: Position
MBR_AN_SAFETY_ZONE_BACKWARDS	1029	The minimum position using the Backwards Button Input Default: 0 (Home Position)	FC3/FC16: Position
Current Cut-Off Limit Inwards **MBR_AN_CURRENT_CUTOFF_LIMIT_INWARDS**	1030	Value in mA Default 4000 mA	FC3/FC16: Overcurrent limit in.
Current Cut-Off Limit Outwards **MBR_AN_CURRENT_CUTOFF_LIMIT_OUTWARDS**	1031	Value in mAmp Default 4000 mA	FC3/FC16: Overcurrent limit out.
MBR_AN_VOLTAGE_LOW_LIMIT	1032	Value in mVolt Default 9000 mV	FC3/FC16: Minimum voltage for activation
MBR_AN_VOLTAGE_HIGH_LIMIT	1033	Value in mVolt Default 29000 mV	FC3/FC16: Maximum voltage for activation

MBR_AN_VOLTAGE_HIGH_LIMIT	1033	Value in mVolt Default 29000 mV	FC3/FC16: Maximum voltage for activation
MBR_AN_HM_MIN_SPEED	1034	Minimum speed for homing/learning In %	FC3/FC16:
MBR_AN_HM_MAX_CURRENT	1035	Max current for homing/learning Value in mAmp	FC3/FC16:
MBR_AN_HM_LIMIT_CHECK_TIME	1036		FC3/FC16:
MBR_AN_HM_TIMEOUT	1037	Max used time for homing/learning In mSec.	FC3/FC16:
MBR_AN_RMP_ACC	1038	Deacceleration time in % of Sec	FC3/FC16:
MBR_AN_RMP_DEC	1039	Acceleration time in % of Sec	FC3/FC16:
MBR_AN_RMP_QDEC	1040	If fault the deacceleration time Will be % of Sec.	FC3/FC16:
MBR_AN_MAX_SPEED	1041		Not for use
MBR_AN_DB_SPEED	1042		Not for use
MBR_AN_CURRENT_CHECK_DELAY	1043		Not for use
MBR_AN_CURRENT_LIMIT_TIME	1044		Not for use
Max Stroke Length **Maximal run-position from home offset** **MBR_AC_MAX_STROKE_LENGTH**	1045	uint32_t Encoder Counts (330 counts per motor revolution) Default: 600	FC3/FC16:
MBR_AN_MAX_SLIP_ERROR	1047	Maximum allowed velocity error per second [steps/s] - 0=disabled	Not for use
MBR_AN_VELOCITY_PID_KP	1048	Velocity controller loop PID proportional gain [x1000]	Not for use
MBR_AN_VELOCITY_PID_KI	1049	Velocity controller loop PID integration gain [x1000]	Not for use
MBR_AN_VELOCITY_PID_KD	1050	Velocity controller loop PID diffential gain [x1000]	Not for use
MBR_AN_POSITIONING_STOP_WINDOW	1051	Position window before stopping [steps]	FC3/FC16:
MBR_AN_POSITIONING_STOP_SPEED	1052	Speed used in position controller when stopping in [%] of max speed	FC3/FC16:

Controller Temperature Limit **MBR_AN_TEMPERATURE_LIMIT**	1053	**RW** int16_t x 0.1 C Disabled: 0 Enabled: ...1500 deci C (150.0) Default: -1	FC3/FC16:
Modbus Address **MBR_AN_MODBUS_ADDRESS**	1054	0..255 Default: 8	FC3/FC16:
Modbus Response Delay **MBR_AN_MODBUS_RESPONS_DELAY**	1055	Additional delay in usec. Max 65.5 msec	FC3/FC16:
Home Position **MBR_AN_HOME_POS**	1056	MbrHomePos_t 1: HOME_POS_RETRACTED - counting positive while moving out 2: HOME_POS_EXTENDED - counting positive while moving in Default: MHP_HOME_POS_RETRACTED	FC3/FC16:

MBR_AN_DEAD_ZONE	1057	For use in SPP mode.	FC3/FC16:
CSP Out Voltage **MBR_AN_POS_VOUT**	1058	V-Out • 1: 5 V (default) • 2: 10 V	FC3/FC16:
MBR_AN_VOLTAGE_CHECK_DELAY	1059		FC3/FC16:
MBR_AN_VOLTAGE_LIMIT_TIME,	1060		FC3/FC16:
Maximum Current Measured **MBR_AN_MAXIMUM_CURRENT_MEASURED**	1061	**RO** Value in mAmp	FC3:
Supply Voltage (Actual V_IN) **MBR_AN_SUPPLY_VOLTAGE**	1062	**RO** Value in centi Volt	FC3:
Controller Temperature Measured **MBR_AN_TEMPERATURE**	1063	**RO** int16_t x 0.1 C	FC3:
Min Controller Temperature Measured **MBR_AN_TEMPERATURE_MIN**	1064	**RO** int16_t x 0.1 C	FC3:
Max Controller Temperature Measured **MBR_AN_TEMPERATURE_MAX**	1065	**RO** int16_t x 0.1 C	FC3:
Total Number of Starts Inwards **MBR_AN_START_INWARDS_TOTAL**	1066	**RO** uint32_t	FC3:
Total Number of Starts Outwards **MBR_AN_START_OUTWARDS_TOTAL**	1068	**RO** uint32_t	FC3:
Total Running Time **MBR_AN_RUN_TIME_TOTAL**	1070	**RO** uint32_t sec the motor has run in total	FC3:
Special Service Register **MBR_AN_SPECIAL_SERVICE_REGISTER**	1072	**RO** Bit 0: (EOS switch, Inwards) Bit 1: (EOS switch, Outwards) Bit 2: Hall A Signal Bit 3: Hall B Signal Bit 14: Position valid 0: OK 1: Invalid, homing needed, other run commands are rejected Bit 15: System Initialized 0: OK 1: Has been initialized	FC3:

Number of Power Fails **MBR_AC_POWER_FAILS_TOTAL**	1073	**RO** uint32_t Total number of following five (5): - MSR_OVERVOLTAGE_DETECTED_ON_V_IN - MSR_UNDERVOLTAGE_DETECTED_ON_V_IN - MSR_CURRENT_CUTOFF_OUT - MSR_CURRENT_CUTOFF_IN - MSR_TEMPERATURE	FC3:
Number of Modbus Messages **MBR_AC_MODBUS_MSG_TOTAL**	1075	**RO** uint32_t	FC3:
Number of Modbus Messages with wrong CRC **MBR_AC_MODBUS_MSG_CRC_ERR_TOTAL**	1077	**RO** uint32_t	FC3:
Total Number of Current Cut-Offs In **MBR_AC_CUTOFF_IN_TOTAL**	1079	**RO** uint32_t	FC3:
Total Number of Current Cut-Offs Out **MBR_AC_CUTOFF_OUT_TOTAL**	1081	**RO** uint32_t	FC3:
Number of Times Running with Actuator Tempera-ture Exceeded **MBR_AC_HIGH_TEMP_TIME_TOTAL**	1083	**RO** uint32_t Seconds The controller does not prohibit operation out of temperature range!!!	FC3:
Reset Statistic and Min/Max Counters **MBR_AC_RESET_STATISTIC**	1085	**WO** Any value "!=0" => Reset (= 0 unless specified) - Boot count - Maximum Current Measured - Min Controller Temperature - Max Controller Temperature - Number of Power Fails - Run time with high temperature - Total Number of Current Cut-Offs in - Total Number of Current Cut-Offs out - Total Number of Starts Inwards - Total Number of Starts Outwards - Total Running Time - Number of Modbus Messages - Number of Modbus Messages with wrong CRC	FC16:

MBR_AX_SAFETY_ZONE_FORWARD	2000	uint32_t The maximum position using the Forward Button Input. This register is used with actuators with a longer stroke length than 16-bit, else MBR_AN_SAFETY_ZONE_FORWARD can be used	
		Valid from V4.15 Default: MBR_AN_MAX_STROKE_LENGTH	
MBR_AX_SAFETY_ZONE_BACKWARDS	2002	uint32_t The minimum position using the Backwards Button Input This register is used with actuators with a longer stroke length than 16-bit, else MBR_AN_SAFETY_ZONE_ BACKWARDS can be used Valid from V4.15 Default: 0 (Home Position)	
MBR_AX_PARALLEL_ARRAY_INFO	2004		

Typical Use Cases

Typical Use Cases In this section further descriptions of how to communicate with the Concens icon Actuators are shown. The examples are typical user scenarios and application solutions. All examples include references to registers, which are described in details. Configuration Before integration into a MODBUS system a few parameters of the actuator have to be checked and eventually changed. This preparation is done by use of the CAS PC too and guarantees that the actuator can execute basic functionality. Further fine-tuning may be required to fulfil system-or application requirements.

Run to target:

Before you move the actuator to any new position you must verify that some general prerequisites are fulfilled. Timing (e.g., when the actuator is still moving), environment conditions and errors might mean that the actuator is in a state where further operation is not possible.

1. Write with function code 16 to register 1020 the new position.
2. Write with function code 16 to register 1023 the value 2 for running to target.
3. Read with function code 3 register 1010 for evaluating stop condition.

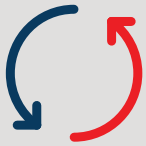
Run to predefined positions:

It is possible to pre-define up to 4 different actuator positions and then have the actuator switch between these positions by simply sending one command.

1. Write with function code 16 to register 1012,1014,1016,1018, the pre-different positions.
2. Write with function code 16 to register 1023 the value 3,4,5 or 6 for running to pre-different position.
3. Read with function code 3 register 1010 for evaluating stop condition.

• NOTRE GAMME DE PRODUITS •

SNT développe son savoir-faire dans tous les domaines
des systèmes de motorisation et d'asservissement.



• MOUVEMENTS ROTATIFS •

- Réducteurs et motoréducteurs
- Renvois d'angle
- Variateurs mécaniques
- Moteurs électriques

• MOUVEMENTS LINÉAIRES •

- Actionneurs linéaires
- Vérins mécaniques
- Unités linéaires
- Vis à billes

• CONTRÔLE DU MOUVEMENT •

- Servo-moteurs brushless et variateurs
- Commandes d'axes
- Variateurs de fréquence
- Pupitres opérateurs
- Variateurs pour moteurs CC

• LIAISONS INTERMÉDIAIRES •

- Accouplements de précision
- Limiteurs de couple
- Barres de liaison

SNT
Le mouvement bien pensé

ZI de la Croix Saint Nicolas - 2, rue Marcel Dassault
94510 La Queue en Brie
Tel: 01 45 93 05 25 - Fax: 01 45 94 79 95
E-mail: contact@snt.tm.fr
www.snt.tm.fr