Motion Control Products

Application note
Integrated Modbus support

AN00198

Rev E (EN)

Modbus is an industry standard protocol that allows a variety of automation devices (such as Programmable Logic Controllers and Human Machine Interfaces) to communicate with each other. The Modbus protocol defines a simple protocol data unit (PDU) that is independent of the underlying communication layers. Inclusion of Modbus protocol support as standard makes integration of ABB products with any other Modbus RTU/TCP device as simple as possible.

Introduction

Controllers communicate (via RS232/422/485 Serial or Ethernet) using a Client / Server topology, in which only one device (the Client) can initiate transactions (called ‘queries’). The other devices (the Servers) respond by supplying the requested data to the Client, or by taking the action requested in the query. Typical Client devices include host processors and programming panels. Typical Servers include programmable controllers.

The Client can address individual Servers, or can initiate a broadcast message to all Servers. Servers return a message (called a ‘response’) to queries that are addressed to them individually. Responses are not returned to broadcast queries from the Client.

The Modbus protocol establishes the format for the Client’s query by placing into it the device (or broadcast) address, a function code defining the requested action, any data to be sent, and an error–checking field. The Server’s response message is also constructed using the Modbus protocol. It contains fields confirming the action taken, any data to be returned, and an error–checking field. If an error occurred in receipt of the message, or if the Server is unable to perform the requested action, the Server will construct an error message and send it as its response.

Methods of Modbus communications configuration

There are two different methods (depending on the motion product used) of configuring communication in the controller/drive via Modbus. Older motion controllers and drives or those with older firmware use a program based Mint interpreter and the newer products have the Modbus functions integrated into the firmware.

Please refer to the table below to select the appropriate firmware for integrated Modbus usage:

Product with integrated Modbus support	**RTU**	**TCP**	Required firmware
NextMove ESB-2	Yes	-	5424 onwards
NextMove e100	Yes	Yes	5633 onwards
MicroFlex e100	Yes	Yes	5633 onwards
MicroFlex e150	Yes	Yes	All versions
MicroFlex e190	No	Yes	All versions
MicroFlex e190 equipped with OPT-SIO-1 option card	Yes	Yes	5900 onwards
MotiFlex e100	Yes	Yes	5633 onwards
MotiFlex e180	No	Yes	All versions

The Integrated Modbus functionality is described in more detail below.

For more specific information on how to use and set up the Mint based Modbus refer to the corresponding Application Note AN00185 - Mint based Modbus RTU Server

Getting Started with Modbus

Data format

When controllers are setup to communicate on a Modbus network using RTU (Remote Terminal Unit) mode, each 8–bit byte in a message contains two 4–bit hexadecimal characters. Each message must be transmitted in a continuous stream.
The Mint interpreter offers support for Client devices with communication settings of:

- 7 data bits, 1 stop bit, Even Parity
- 7 data bits, 1 stop bit, Odd Parity
- 8 data bits, 1 stop bit, No Parity

Baud rates are limited to those supported by the Mint controller (and of course must match the baud rate setting made on the Modbus Client device).

Message framing

In RTU mode, messages start with a silent interval of at least 3.5 character times. This is most easily implemented as a multiple of character times at the baud rate that is being used on the network.

The first field transmitted is the device address (in this case the node address of the Mint controller). Networked devices monitor the network bus continuously, including during the 'silent' intervals. When the first field (the address field) is received, each device decodes it to find out if it is the addressed device.

Following the last transmitted character, an interval of at least 3.5 character times marks the end of the message. A new message can begin after this interval. The entire message frame must be transmitted as a continuous stream. If a silent interval of more than 1.5 character times occurs before completion of the frame, the receiving device flushes the incomplete message and assumes that the next byte will be the address field of a new message.

A typical message frame is shown below.

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
	8 bits	8 bits	n * 8 bits	16 bits	
{delay}	[STX]	[ETX]	[NUL][SOH][NUL][SOH]	Õ ù	{delay}

ADDRESS
8 bits
[STX]

The **address** field of a RTU message frame contains one character (8 bits). Valid Server device addresses are in the range of 0 – 255 decimal.

When the Server sends its response, it places its own address in the address field of the response to let the Client know which Server is responding. Address 0 is used for the broadcast address, which all Server devices recognize.

When configuring a Mint controller for use as a Modbus Server it is therefore advisable to avoid setting this controller up as Mint Node 0 (note that the Client can only use the broadcast address for write transactions).

FUNCTION
8 bits
[ETX]

The **function** code field of a RTU message frame contains one character (8 bits). Valid codes (from the Client) are in the range of 1 – 127 decimal.

When a message is sent from a Client to a Server device the function code field tells the Server what kind of action to perform. Examples are to read the ON/OFF states of a group of discrete inputs; to read the data contents of a group of registers; to read the diagnostic status of the Server or to write to designated registers.

When the Server responds to the Client, it uses the function code field to indicate either a normal (error–free) response or that some kind of error occurred (called an exception response). For a normal response, the Server simply echoes the original function code. For an exception response, the Server returns a code that is equivalent to the original function code with its most–significant bit set to a logic 1.

DATA
n * 8 bits
[NUL][SOH][NUL][SOH]

The **data** field is constructed using sets of two hexadecimal digits, in the range of 00 to FF hexadecimal that are then packed into a single ASCII character. The data field of messages sent from a Client to Server devices contains additional information which the Server must use to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

For example, if the Client requests a Server to read a group of holding registers (function code 03), the data field specifies the starting register and how many registers are to be read. If the Client writes to a group of registers in the Server (function code 16 decimal), the data field specifies the starting register, how many registers to write, the count of data bytes to follow in the data field, and the data to be written into the registers.

If no error occurs, the data field of a response from a Server to a Client contains the data requested. If an error occurs, the field contains an exception code that the Client application can use to determine the next action to be taken.

CRC CHECK
16 bits
Õ ù

When RTU mode is used for character framing, the error checking field contains two ASCII characters. The error check characters are the result of a **Cyclic Redundancy Check** (CRC) calculation that is performed on the message contents.

The CRC characters are appended to the message as the last field.

Exception responses

When a Client device sends a query to a Server device it expects a normal response. One of four possible events can occur from the Client's query:

- If the Server device receives the query without a communication error, and can handle the query normally, it returns a normal response.
- If the Server does not receive the query due to a communication error, no response is returned. The Client program will eventually process a timeout condition for the query.
- If the Server receives the query, but detects a communication error (incorrect CRC), no response is returned. The Client program will eventually process a timeout condition for the query.
- If the Server receives the query without a communication error, but cannot handle it (for example, if the request is to read a non–existent register), the Server will return an exception response informing the Client of the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

Function code field: In a normal response, the Server echoes the function code of the original query in the function code field of the response. All function codes have a most–significant bit (MSB) of 0 (their values are all below 80 hexadecimal).

In an exception response, the Server sets the MSB of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response.

With the function code's MSB set, the Client's application program can recognize the exception response and can examine the data field for the exception code.

Data field: In a normal response, the Server may return data or statistics in the data field (any information that was requested in the query). In an exception response, the Server returns an exception code in the data field. This defines the Server condition that caused the exception.

An illegal function response is generated if the Client requests a function other than those supported by the interpreter.

Mint specific settings

MicroFlex e190 and MotiFlex e180 are capable of operating as Modbus Clients and/or Servers via Modbus TCP. MicroFlex e190 can be equipped with a OPT-SIO-1 option card and is also capable of operating as Modbus Server via Modbus RTU.

NextMove e100 and ESB-2 are only capable of operating as Modbus servers. Both products support Modbus RTU, only NextMove e100 supports Modbus TCP

Support is provided for the following Modbus Server functions:

- 03 – Read holding registers
- 04 – Read input registers
- 06 – Preset single register
- 16 – Preset multiple registers
- 23 – Read / write 4x registers

Support is provided for the following Modbus Client functions (only MicroFlex e190 and MotiFlex e180):

- 03 – Read holding registers
- 04 – Read input registers
- 16 – Preset multiple registers

Depending on the product, there are two different data areas available that are used for data exchange:

- NetData array
- Comms array

The following table shows the use of the different data areas:

Drive/Controller	Modbus Type	Data area
MicroFlex e190	Modbus TCP Client/Server	NetData
MicroFlex e190 + OPT-SIO-1	Modbus RTU Server	Comms
MotiFlex e180	Modbus TCP Client/Server	NetData
NextMove ESB-2	Modbus RTU	Comms
e100 products	Modbus RTU/TCP	Comms / NetData (see MODBUSPARAMETER below)

For e100 products we can choose where the data goes set by the Mint keyword MODBUSPARAMETER:

NetData array:
MODBUSPARAMETER(_busSERIAL1, _mpREGISTER_MAPPING) = _rmNET_DATA

Comms array:
MODBUSPARAMETER(_busSERIAL1, _mpREGISTER_MAPPING) = rmCOMMS_ARRAY

This data area has a fixed mapping with respect to how it appears to a Client device as Modbus registers as shown by the table below (equivalent AC500 addresses are also shown for reference):

Server Modbus register	AC500 address		Mint Comms array (Comms=Real, Commsinteger = DWord)		Mint Netdata array (Netfloat = Real, Netinteger = DWord)	
0	%MW0.0	%MD0.0	Invalid	Invalid	Element 0 LSW	Element 0
1	%MW0.1		Invalid		Element 0 MSW	
2	%MW0.2	%MD0.1	Element 1 LSW	Element 1	Element 1 LSW	Element 1
3	%MW0.3		Element 1 MSW		Element 1 MSW	
4	%MW0.4	%MD0.2	Element 2 LSW	Element 2	Element 2 LSW	Element 2
5	%MW0.5		Element 2 MSW		Element 2 MSW	
…	---	---	---	---	---	---
198	%MW0.198	%MD0.99	Element 99 LSW	Element 99	Element 99 LSW	Element 99
199	%MW0.199		Element 99 MSW		Element 99 MSW	
200	%MW0.200	%MD0.100	Invalid	Invalid	Element 100 LSW	Element 100
201	%MW0.201		Invalid		Element 100 MSW	
202	%MW0.202	%MD0.101	Invalid	Invalid	Element 101 LSW	Element 101
203	%MW0.203		Invalid		Element 101 MSW	
…	---	---	---	---	---	---
1996	%MW0.1996	%MD0.998	Invalid	Invalid	Element 998 LSW	Element 998
1997	%MW0.1997		Invalid		Element 998 MSW	
1998	%MW0.1998	%MD0.999	Invalid	Invalid	Element 999 LSW	Element 999
1999	%MW0.1999		Invalid		Element 999 MSW	

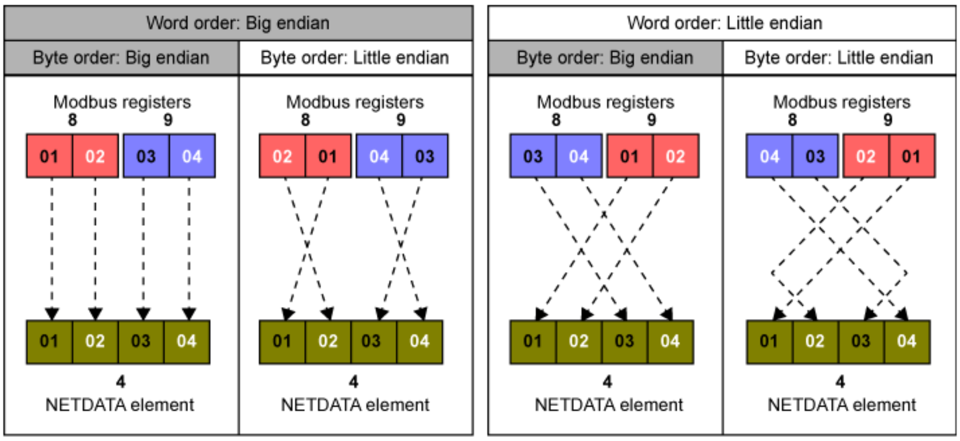
LSW – Least Significant Word : MSW – Most Significant Word

The Mint Comms array provides 99 elements (1-99). If the Client attempts to access Comms element 0 or Comms elements greater than 99 the Mint controller will return the ‘Invalid Data Address’ Modbus exception packet.

The Mint Netdata array (not supported on NextMove ESB-2) provides 1000 elements (0-999). If the Client attempts to access Netdata elements greater than 999 the e190/e180 will return the ‘Invalid Data Address’ Modbus exception packet.

Registers are addressed starting at zero. However Mint Comms locations are addressed from 1 – 99 (see also Application Note AN00110) so care should be taken not to try to read or write register 0 on the Mint controller if using Comms instead of NVLONG.

When using Modbus the byte and word order of the Modbus data is important. It is possible to change the settings for both byte and word order for Client and Server operation in the drives. The default byte and word orders are configured to match the requirements for communication with other ABB Motion, PLC and HMI products. Some third party Modbus products tend to use little endian word order so when using a third party Client it may be necessary to adjust this setting in the drive.

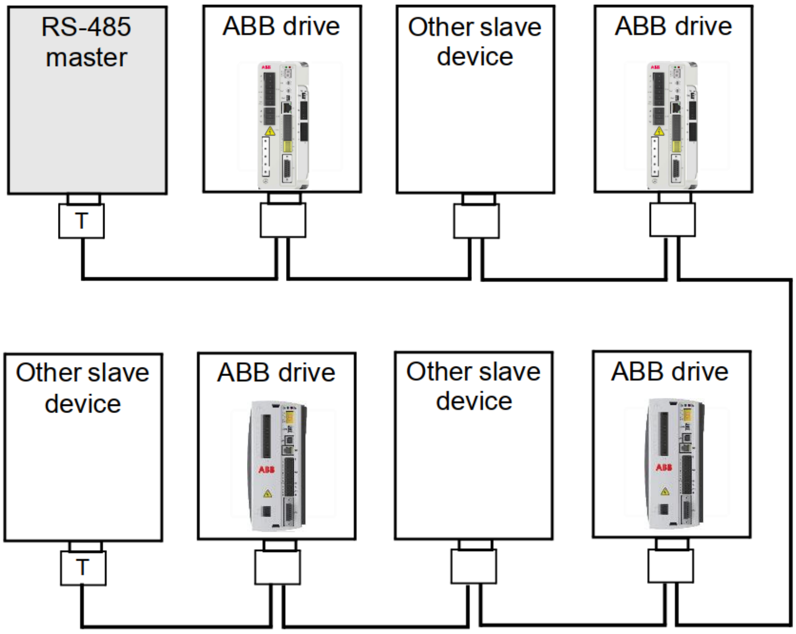


Modbus RTU and TCP are also supported as standard on the CP600 and CP600-eco ranges of HMI panels as well as the AC500 PLC range.

Network topology

Modbus RTU

When using 2-wire RS485 be sure to include 120 ohm terminating resistors between data lines A and B at each end of the network to avoid data corruption.

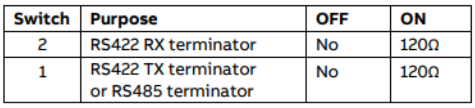


T = Termination

When using 4-wire RS422 be sure to include 120 ohm terminating resistors between RX+ and RX- at each end of the network to avoid data corruption.

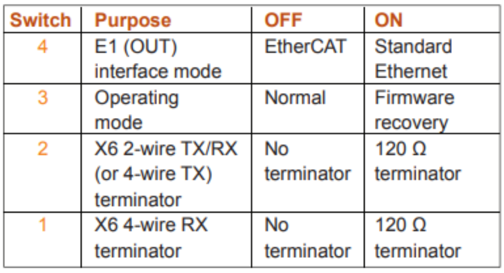
When using MicroFlex e190 with OPT-SIO-1, the termination could be realized with the dip switches on the option module.
When using MicroFlex e150, the termination could be realized with the dip switches at the front side of the drive
No external resistor is needed.

MicroFlex e190



Switch	Purpose	OFF	ON
2	RS422 RX terminator	No	120Ω
1	RS422 TX terminator or RS485 terminator	No	120Ω

MicroFlex e150



Switch	Purpose	OFF	ON
4	E1 (OUT) interface mode	EtherCAT	Standard Ethernet
3	Operating mode	Normal	Firmware recovery
2	X6 2-wire TX/RX (or 4-wire TX) terminator	No terminator	120 Ω terminator
1	X6 4-wire RX terminator	No terminator	120 Ω terminator

For MicroFlex e100 and NextMove, the termination must be realized externally.

Modbus TCP

When using Modbus TCP / Ethernet, wire the network in a ‘star’ type configuration using an Ethernet switch between the Client and the connected Server(s). Straight-through or crossover cables may be used between the devices and the switch. If the controller is part of an Ethernet Powerlink (EPL) network then it is also necessary to include an EPL Router (part: OPT036-501) or a MicroFlex e190 / MotiFlex e180 drive acting as an EPL router between the switch and the devices (see Application Note AN00247 for further details).

Note: AC500 and CP600 products use a non-standard RS232 pinout so be sure to check the relevant product manual before connecting other serial devices to these products. Failure to observe the correct pinout may result in damage to the connected device.

Supported connection types

Controller

The table below shows the physical connection possibilities for NextMove products supporting integrated Modbus protocols. AC500 and CP600 products are included for reference.

Connection Type	NextMove e100	NextMove ESB-2	AC500	AC500 Eco	CP600
RS232	Yes	Yes (by variant)	Yes	No	Yes
2 wire RS485	Yes	No	Yes	Yes	Yes
4 wire RS422	Yes	Yes (by variant)	No	No	Yes
Ethernet	Yes	No	Yes	Yes (by variant)	Yes

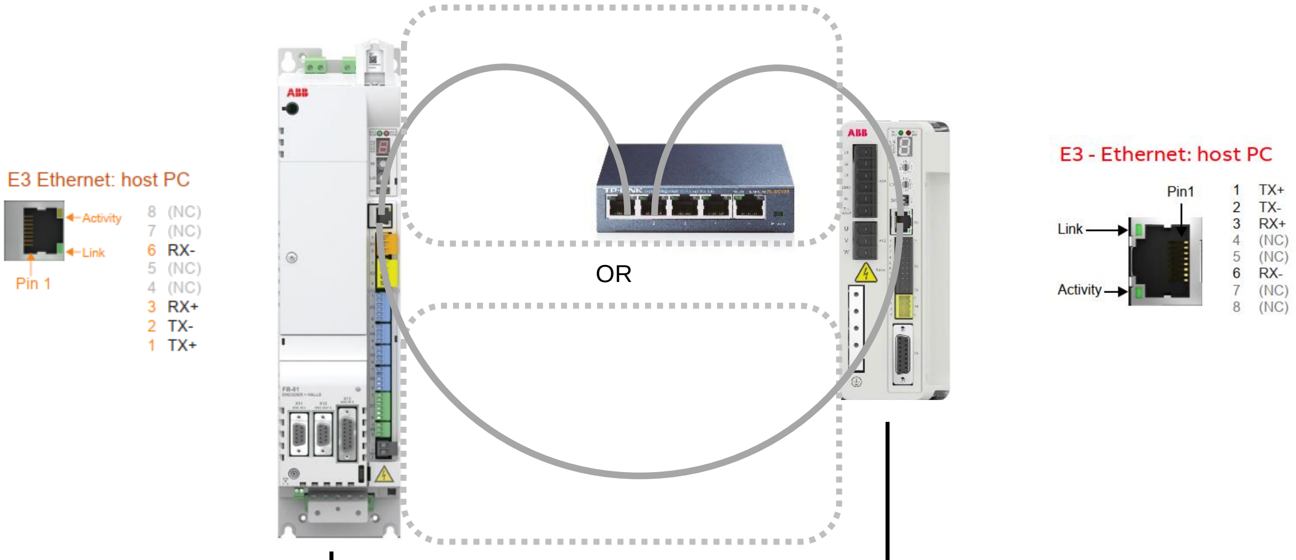
Drives

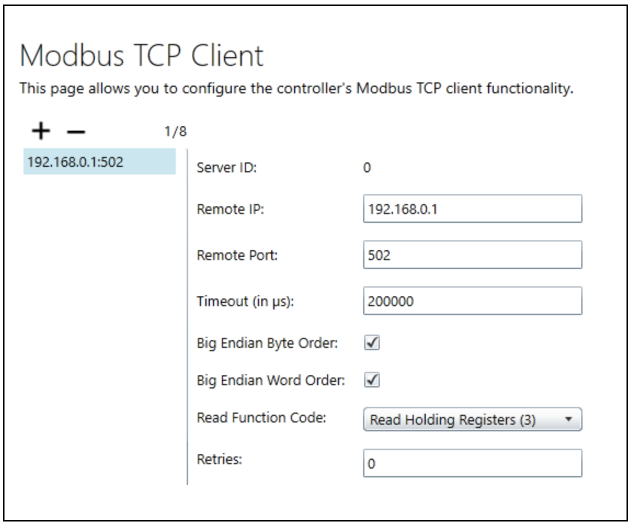
The table below shows the physical connection possibilities for MicroFlex e1x0 and MotiFlex e180 drive products supporting integrated Modbus protocols.

Connection Type	MicroFlex e100	MicroFlex e150	MicroFlex e190	MotiFlex e180
RS232	No	No	No	No
2 wire RS485	Yes	Yes	Yes (OPT-SIO-1)	No
4 wire RS422	No	Yes	Yes (OPT-SIO-1)	No
Ethernet	Yes	Yes	Yes	Yes

Example 1:

Connect MotiFlex e180 (Modbus TCP Client) to MicroFlex e190 (Modbus TCP Server) via Switch or via crossover cable





Modbus TCP Client

This page allows you to configure the controller's Modbus TCP client functionality.

+ − 1/8

192.168.0.1:502

Server ID: 0

Remote IP: 192.168.0.1

Remote Port: 502

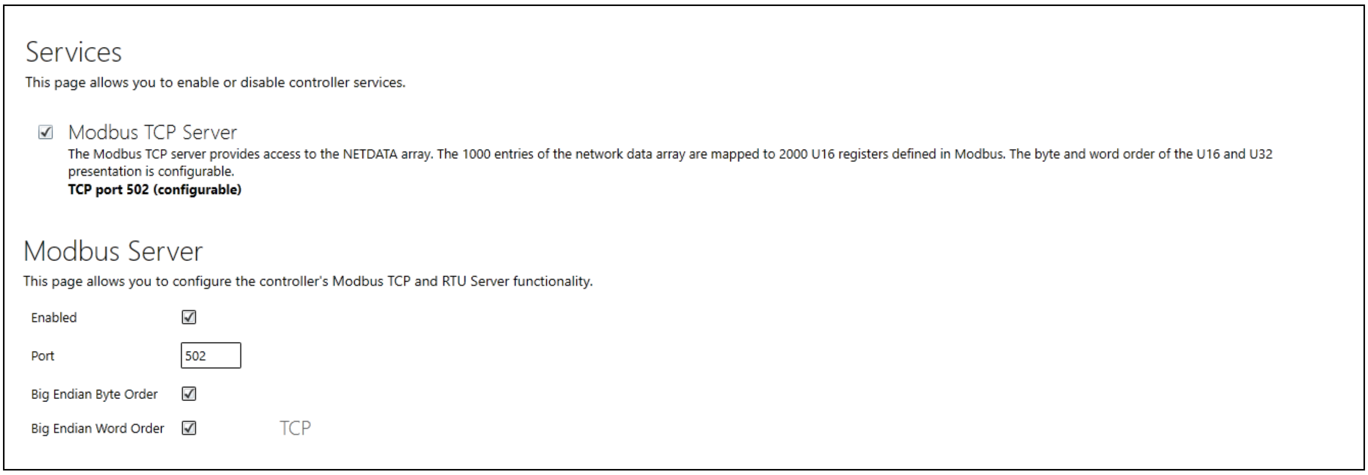
Timeout (in µs): 200000

Big Endian Byte Order: ☑

Big Endian Word Order: ☑

Read Function Code: Read Holding Registers (3)

Retries: 0



Services

This page allows you to enable or disable controller services.

☑ Modbus TCP Server

The Modbus TCP server provides access to the NETDATA array. The 1000 entries of the network data array are mapped to 2000 U16 registers defined in Modbus. The byte and word order of the U16 and U32 presentation is configurable.

TCP port 502 (configurable)

Modbus Server

This page allows you to configure the controller's Modbus TCP and RTU Server functionality.

Enabled ☑

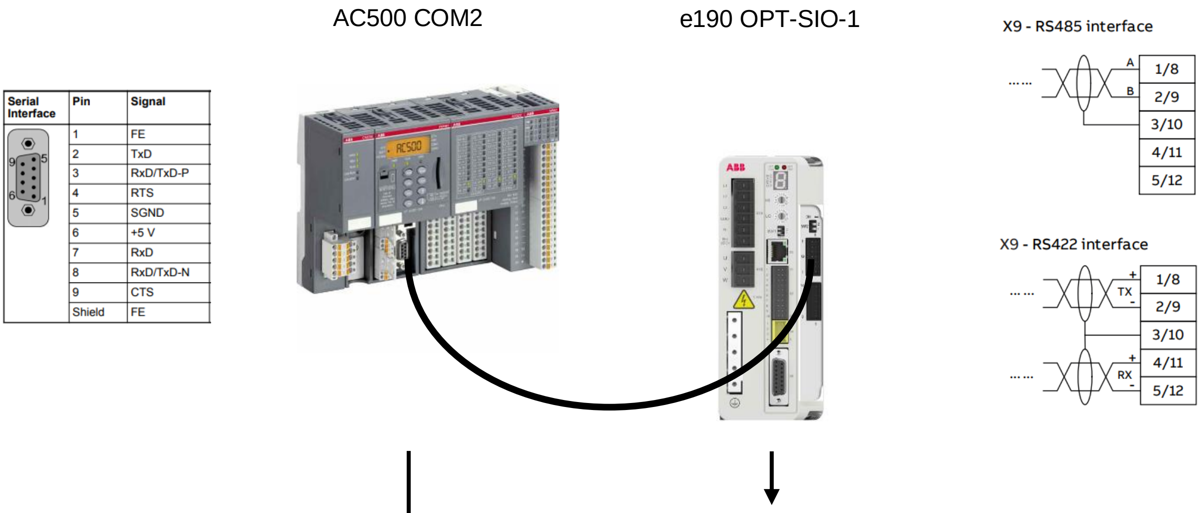
Port 502

Big Endian Byte Order ☑

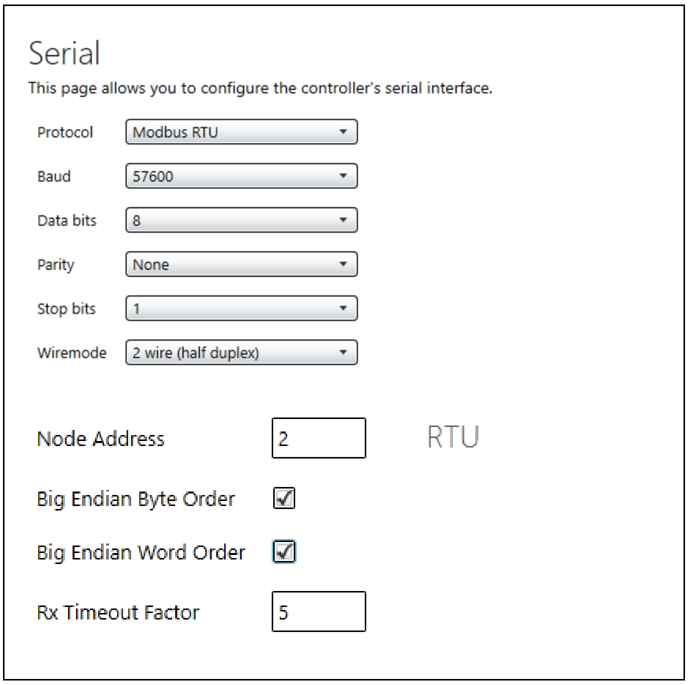
Big Endian Word Order ☑ TCP

Example 2:

Connect MicroFlex e190 equipped with OPT-SIO-1 option card (Modbus RTU Server) to an external PLC (e.g. AC500 COM2)
Please refer to the OPT-SIO-1 Option Card of MicroFlex e190 Servo Drive Quick installation guide for more information about how to install the option card.



Serial Interface	Pin	Signal
	1	FE
	2	TxD
	3	RxD/TxD-P
	4	RTS
	5	SGND
	6	+5 V
	7	RxD
	8	RxD/TxD-N
	9	CTS
	Shield	FE

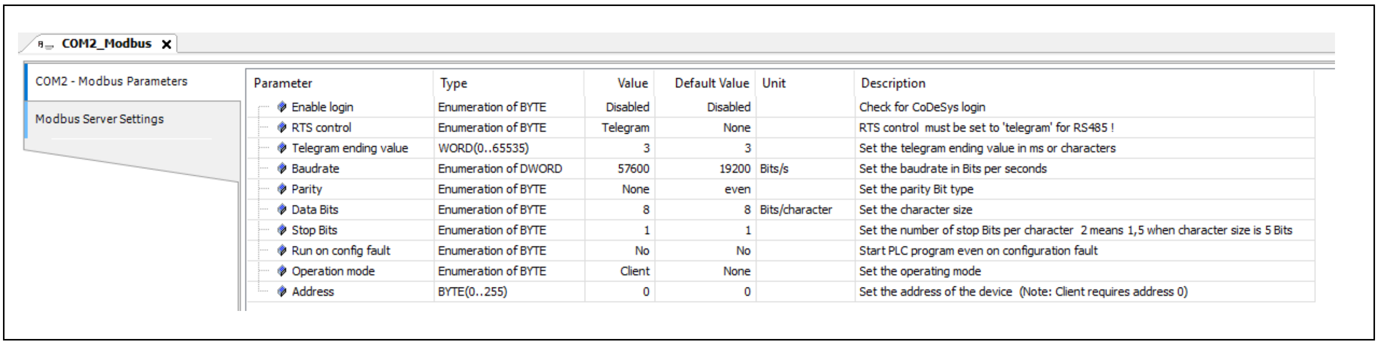


Serial

This page allows you to configure the controller's serial interface.

Protocol: Modbus RTU
Baud: 57600
Data bits: 8
Parity: None
Stop bits: 1
Wiremode: 2 wire (half duplex)

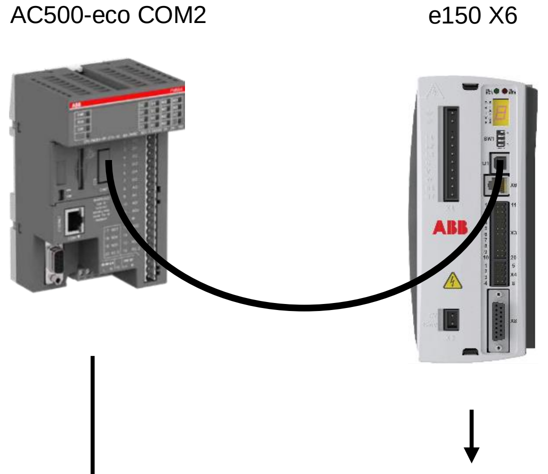
Node Address: 2 RTU
Big Endian Byte Order ☑
Big Endian Word Order ☑
Rx Timeout Factor: 5



Parameter	Type	Value	Default Value	Unit	Description
Enable login	Enumeration of BYTE	Disabled	Disabled		Check for CoDeSys login
RTS control	Enumeration of BYTE	Telegram	None		RTS control must be set to 'telegram' for RS485 !
Telegram ending value	WORD(0..65535)	3	3		Set the telegram ending value in ms or characters
Baudrate	Enumeration of DWORD	57600	19200	Bits/s	Set the baudrate in Bits per seconds
Parity	Enumeration of BYTE	None	even		Set the parity Bit type
Data Bits	Enumeration of BYTE	8	8	Bits/character	Set the character size
Stop Bits	Enumeration of BYTE	1	1		Set the number of stop Bits per character 2 means 1,5 when character size is 5 Bits
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Operation mode	Enumeration of BYTE	Client	None		Set the operating mode
Address	BYTE(0..255)	0	0		Set the address of the device (Note: Client requires address 0)

Example 3:

Connect MicroFlex e150 (Modbus RTU Server) to an external PLC (e.g. AC500-eco)

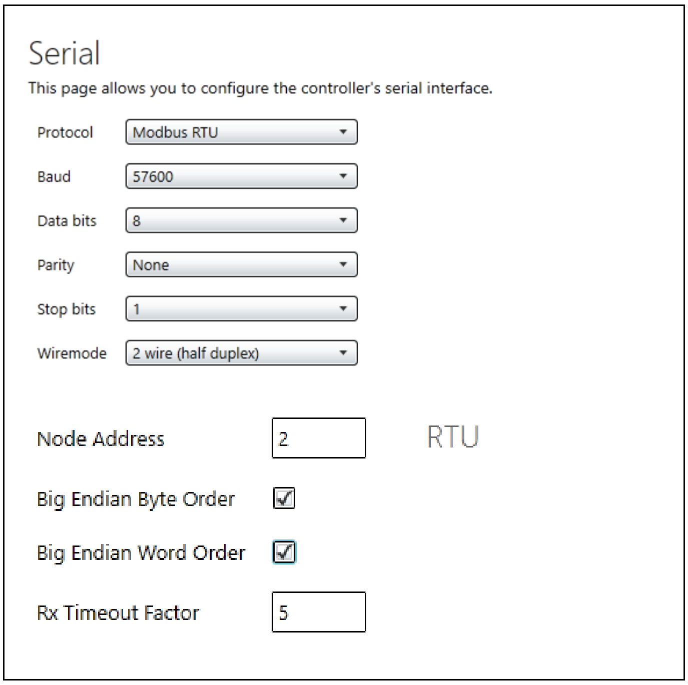


⑨ **COM2**

1	Terminator P
2	TxD/RxD-P
3	TxD/RxD-N
4	Terminator N
5	Functional earth

X6 - RS485 interface

	2-wire	4-wire
1	TXA(+)/RXA(+)	TXA(+)
2	TXB(-)/RXB(-)	TXB(-)
3	GND	GND
4	+7 V out	+7 V out
5	(NC)	RXA(+)
6	(NC)	RXB(-)



Serial

This page allows you to configure the controller's serial interface.

Protocol: Modbus RTU
Baud: 57600
Data bits: 8
Parity: None
Stop bits: 1
Wiremode: 2 wire (half duplex)

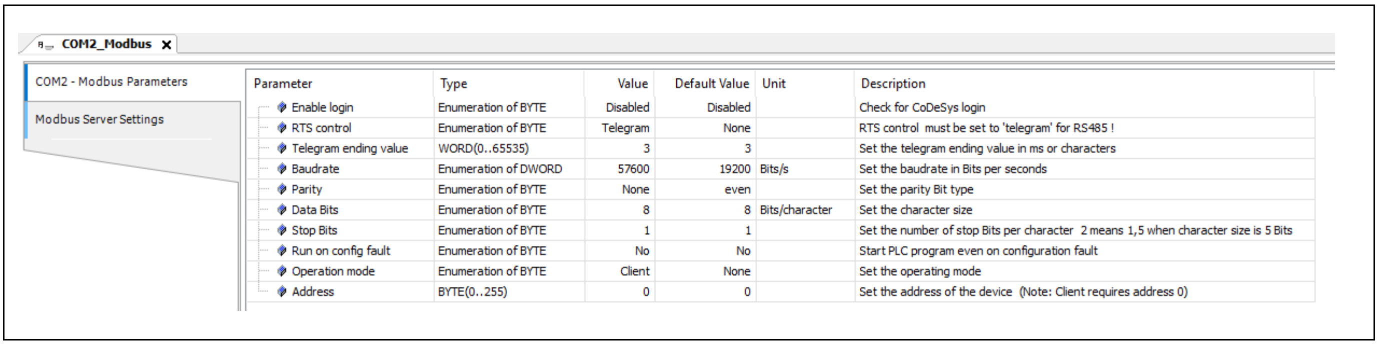
Node Address: 2

RTU

Big Endian Byte Order ☑

Big Endian Word Order ☑

Rx Timeout Factor: 5



Parameter	Type	Value	Default Value	Unit	Description
Enable login	Enumeration of BYTE	Disabled	Disabled		Check for CoDeSys login
RTS control	Enumeration of BYTE	Telegram	None		RTS control must be set to 'telegram' for RS485 !
Telegram ending value	WORD(0..65535)	3	3		Set the telegram ending value in ms or characters
Baudrate	Enumeration of DWORD	57600	19200	Bits/s	Set the baudrate in Bits per seconds
Parity	Enumeration of BYTE	None	even		Set the parity Bit type
Data Bits	Enumeration of BYTE	8	8	Bits/character	Set the character size
Stop Bits	Enumeration of BYTE	1	1		Set the number of stop Bits per character 2 means 1,5 when character size is 5 Bits
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Operation mode	Enumeration of BYTE	Client	None		Set the operating mode
Address	BYTE(0..255)	0	0		Set the address of the device (Note: Client requires address 0)

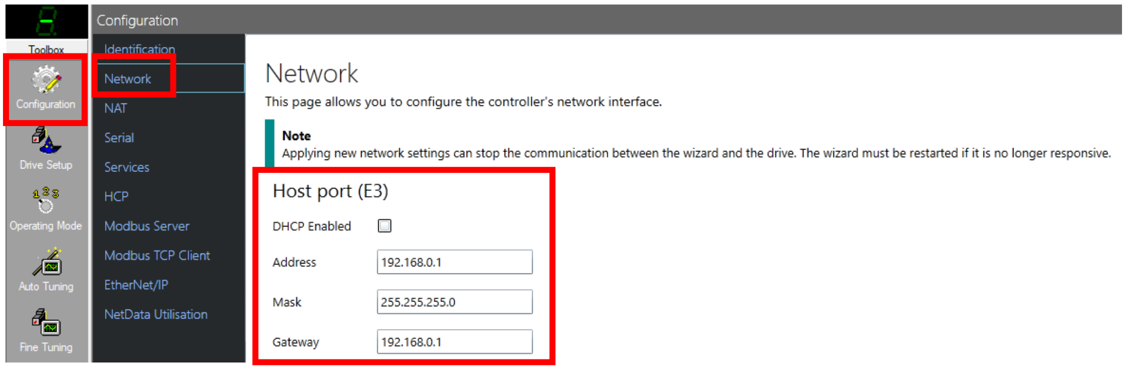
MicroFlex e190 and MotiFlex e180 Modbus configuration

MicroFlex e190 and MotiFlex e180 Modbus operation is configured via the ‘Configuration’ screens within Mint Workbench when online to the drive. All other products use Mint keywords to enable and configure Modbus Server operation. Download firmware from new.abb.com/motion or contact cn-motionsupport@cn.abb.com to obtain the relevant Mint System File (.msx) to update a drive / motion controller. Note that Modbus support is only offered on Revision B e100 products (e.g. NXE100B-1608D**B**W).

Configuration (Modbus TCP)

All Modbus configuration for the MicroFlex e190 and MotiFlex e180 drives is performed via the ‘Configuration’ screens within Mint Workbench (these configuration settings can be saved as part of a controller archive to be transferred to another drive if necessary). Once online to the drive click on the ‘Configuration’ button in the Workbench Toolbox to display this work area. Note that for this screen to operate correctly the e190/e180 must have been ‘Discovered’ via the Mint HTTP Server (please refer to the MicroFlex e190 or MotiFlex e180 Installation Manual for further details about the operation of the Mint HTTP Server and the use of ‘Discovery’ to automatically scan and detect these devices).

Click on ‘Network’ to configure the drive’s IP address (Modbus TCP uses the ‘Host port’ connection on the drive which is numbered E3)…



Network

This page allows you to configure the controller's network interface.

Note
Applying new network settings can stop the communication between the wizard and the drive. The wizard must be restarted if it is no longer responsive.

Host port (E3)

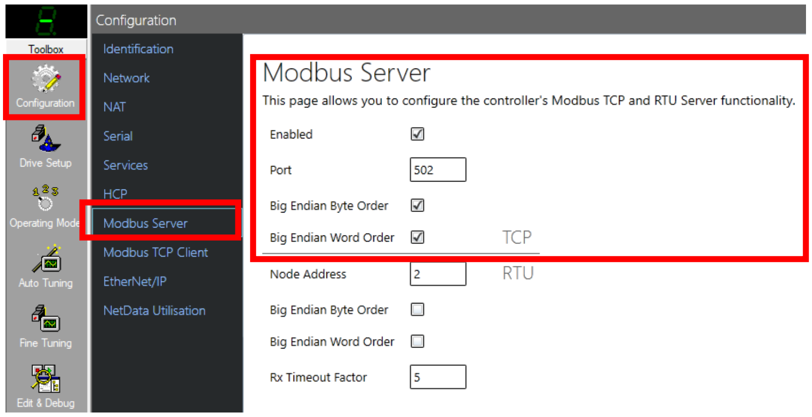
DHCP Enabled □

Address 192.168.0.1

Mask 255.255.255.0

Gateway 192.168.0.1

The Gateway address must be on the same subnet as the drive (e.g. if the drive is 192.168.0.1 then the Gateway must be 192.168.0.x where x is often just set to an unused address). Click on the ‘Apply’ button at the bottom of the screen to make any changes. Click on ‘Modbus TCP Server’ to configure Server operation…



Modbus Server

This page allows you to configure the controller's Modbus TCP and RTU Server functionality.

Enabled ☑

Port 502

Big Endian Byte Order ☑

Big Endian Word Order ☑

TCP

Node Address 2

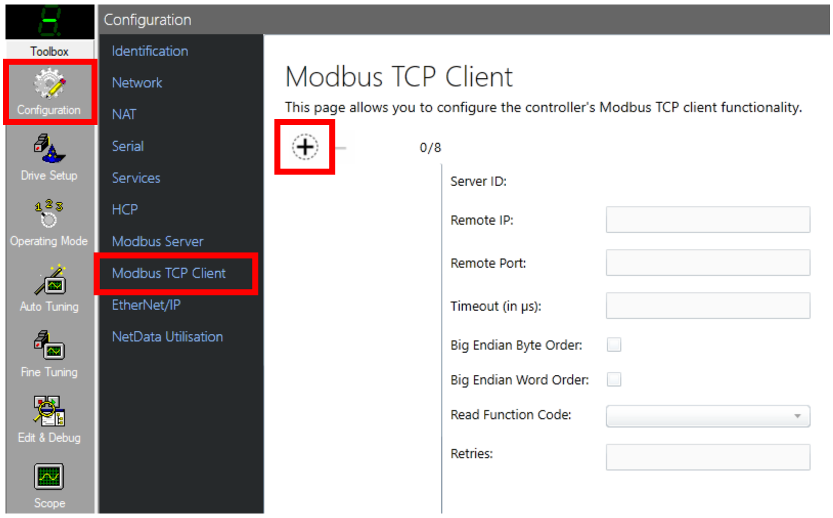
RTU

Big Endian Byte Order □

Big Endian Word Order □

Rx Timeout Factor 5

Check boxes are provided to enable/disable Server operation (enabled by default) and to configure the byte and word order of the Modbus data (set to Big Endian by default for compatibility with other ABB products supporting Modbus TCP operation). The standard Modbus port number is 502, but this can be changed if required to suit third party products using a custom port number.



Modbus TCP Client

This page allows you to configure the controller's Modbus TCP client functionality.

0/8

Server ID:

Remote IP:

Remote Port:

Timeout (in µs):

Big Endian Byte Order: □

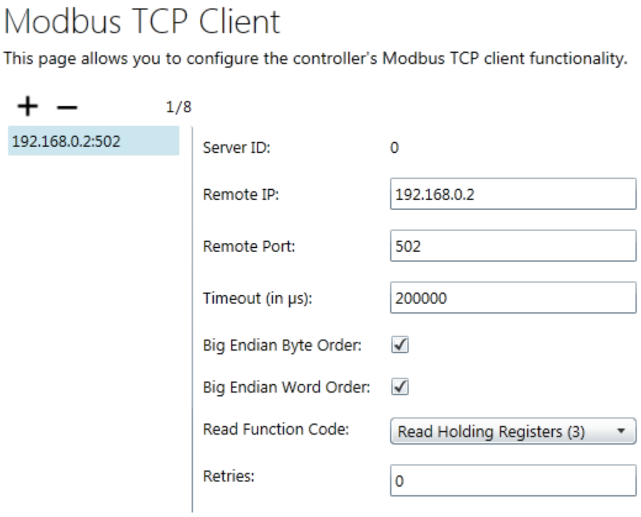
Big Endian Word Order: □

Read Function Code:

Retries:

Click on the 'Apply' button at the bottom of the screen to make any changes. Click on 'Modbus TCP Client' to configure a list of connected Server devices…

Initially the list of connected devices is empty, so click on the '+' button to add a remote (Server) device. The Workbench will automatically add some default values as shown below…


Modbus TCP Client

This page allows you to configure the controller's Modbus TCP client functionality.

1/8

192.168.0.2:502

Server ID: 0

Remote IP: 192.168.0.2

Remote Port: 502

Timeout (in µs): 200000

Big Endian Byte Order: ☑

Big Endian Word Order: ☑

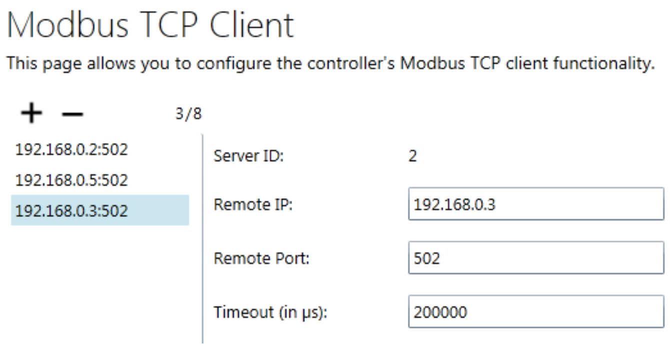
Read Function Code: Read Holding Registers (3)

Retries: 0

These settings can be edited to suit the configuration of the connected Modbus TCP Server devices.

It is important to note the 'Server ID' associated with the remote device – this is automatically allocated and is one of the parameters used by the MODBUSTCPxx Mint keywords to read/write data on the Server device (so for example, if Server ID 0 relates to 192.168.0.2 as shown above, we would use MODBUSTCP32(0,0) to access Netdata 0 on a MicroFlex e190 or MotiFlex e180 configured with IP address 192.168.0.2 and connected to the Client drive).

The list below the + and – buttons is ordered in Server ID order (with 0 at the top) rather than IP address order...



Modbus TCP Client

This page allows you to configure the controller's Modbus TCP client functionality.

Server ID: 2

Remote IP: 192.168.0.3

Remote Port: 502

Timeout (in µs): 200000

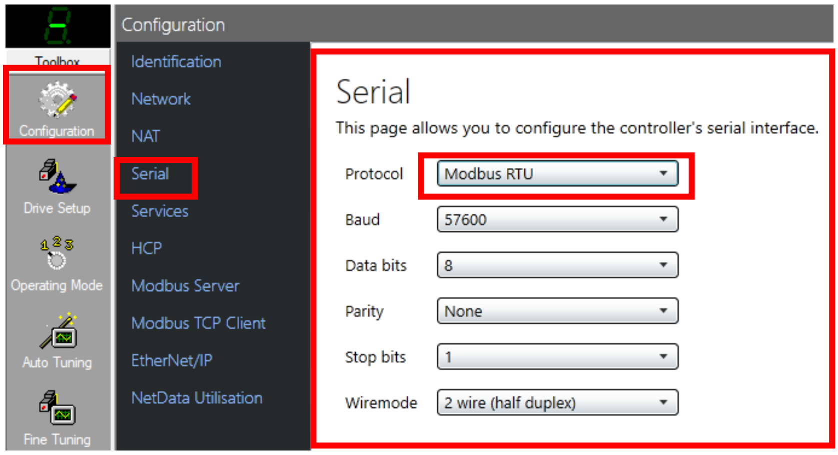
Note that all devices on the TCP network must have unique IP addresses (so if one of the Server devices is 192.168.0.1 this address cannot be used in the 'Network' settings for the Client drive).

By default the number of retries for Modbus Client transactions is set to 0 (no retries). This should be increased if automatically retrying Modbus transactions is preferable to handling an error in a Mint ONERROR event (the larger the number of connected devices the more likely that reties will be necessary).

Click on the 'Apply' button at the bottom of the screen to make any changes.

Configuration (Modbus RTU)

Also the Modbus RTU configuration for the MicroFlex e190 drives is performed via the 'Configuration' screens within Mint Workbench (these configuration settings can be saved as part of a controller archive to be transferred to another drive if necessary). Once online to the drive click on the 'Configuration' button in the Workbench Toolbox to display this work area. Note that for this screen to operate correctly the e190 must have been 'Discovered' via the Mint HTTP Server (please refer to the MicroFlex e190 Installation Manual for further details about the operation of the Mint HTTP Server and the use of 'Discovery' to automatically scan and detect these devices).



Serial

This page allows you to configure the controller's serial interface.

Protocol: Modbus RTU

Baud: 57600

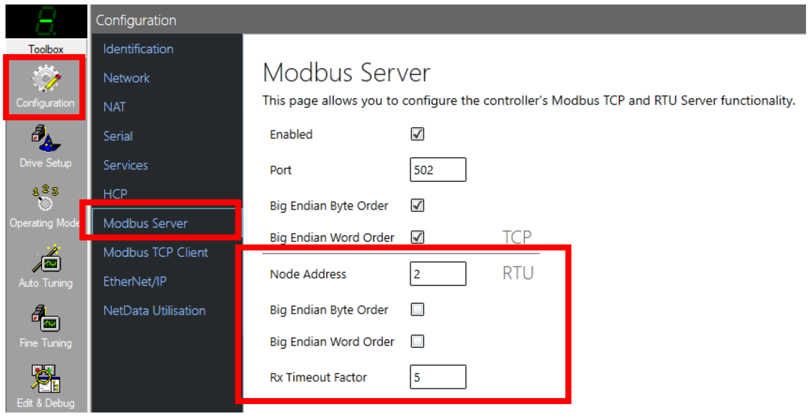
Data bits: 8

Parity: None

Stop bits: 1

Wiremode: 2 wire (half duplex)

Click on 'Serial' to configure the drive's RS422/485 serial port settings. (Modbus RTU must be activated by selecting "Modbus RTU" as Protocol) Adjust all settings according to the Client settings.



Modbus Server

This page allows you to configure the controller's Modbus TCP and RTU Server functionality.

Enabled ☑

Port 502

Big Endian Byte Order ☑

Big Endian Word Order ☑ TCP

Node Address 2 RTU

Big Endian Byte Order □

Big Endian Word Order □

Rx Timeout Factor 5

Check boxes are provided to configure the byte and word order of the Modbus data (set to Big Endian by default for compatibility with other ABB products supporting Modbus RTU operation).

Note: Support is included for broadcast write functions (i.e. to node ID 0) so node ID 0 should be avoided unless broadcast functions are intended.

Example Mint Code

For operation as a Modbus RTU/TCP Server no Mint code is required. The Client device will access Netdata via the Modbus register mappings detailed in the previous section.

The configuration process allows the user to define whether Modbus function code 03 or 04 is used by the Client when reading data from another Modbus TCP device. The Modbus TCP Client always uses function code 16 to write to Modbus data on a connected Server device (even when accessing a single register).

For operation as a Modbus TCP Client the Mint keywords MODBUSTCP16 (to access a single 16 bit Modbus register), MODBUSTCP32 (to access a pair of 16 bit Modbus registers) and MODBUSTCPFLOAT (to access a single 32 bit floating point Modbus register) are used.

Example 1:

Client e180 wishes to take data from a local integer variable and write this to Netinteger(3) on a MicroFlex e190 via Modbus TCP. The MicroFlex e190 has IP address 192.168.0.11 and is configured as Server ID 4 in the list of connected devices…

```
Dim nValue As Integer
MODBUSTCP32 (4, 6) = nValue
```

Example 2:

Client e180 wishes to read data from a remote NETFLOAT location 2 on another MotiFlex e180 via Modbus TCP. The MotiFlex e180 (Server) has IP address 192.168.0.2 and is configured as Server ID 1 in the list of connected devices…

```
Dim fValue As Float
NETINTEGER(999) = MODBUSTCP32 (1, 4)
fValue = NETFLOAT(999)
```

Because all Modbus data is transferred as an integer note the use of a local (unused) Netdata location (999) in this example to perform the “conversion” between a 32 bit integer representation of the floating point value (IEEE format) and the actual floating point value stored in the variable ‘fValue’.
However, to avoid this additional processing a dedicated Modbus Client function, MODBUSTCPFLOAT is provided. This allows the user to read 32-bit floating point data from a Server and transfer this directly to local data. Using the same example above the code using MODBUSTCPFLOAT becomes…

```
Dim fValue As Float
fValue = MODBUSTCPFLOAT(1,4)
```

Example 3:
Client e190 wishes to write an integer value of 3 to a remote Modbus register (location 0x3B00) on a vision system supporting operation as a Modbus TCP Server. The vision system has IP address 192.168.0.3 and is configured as Server ID 2 in the list of connected devices…

```
MODBUSTCP16 (2, 0x3B00) = Int16(0x0003)
```

```
Function Int16(ByVal a As Integer) As Integer
    Int16 = IIf(a > 0x7FFF, a - 0x10000, a)
End Function
```

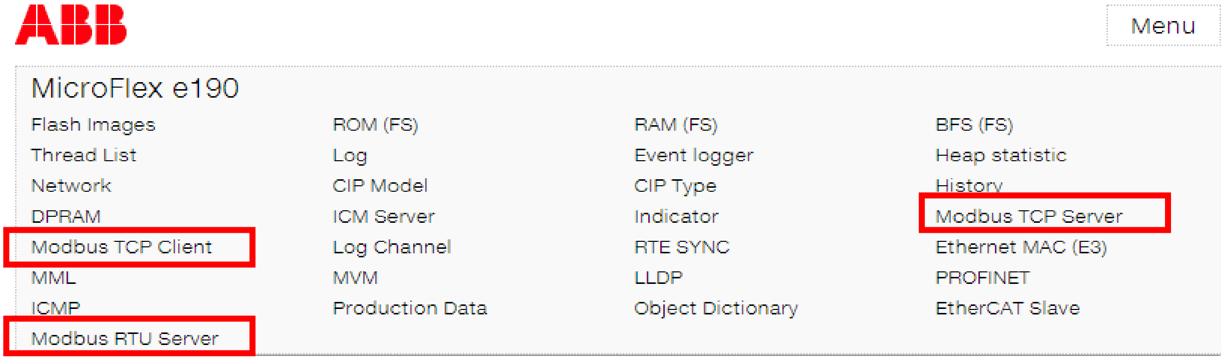
Note the use of the custom Int16 function written in Mint to ensure the correct representation of signed 16 bit data.

Example 4:
Client e190 wishes to write a floating point value of 1.234 to NETFLOAT(1) on a remote MotiFlex e180 connected via Modbus TCP. The MotiFlex e180 has IP address 192.168.0.2 and is configured as Server ID 0 in the list of connected devices…

```
MODBUSTCPFLOAT(0,2) = 1.234
```

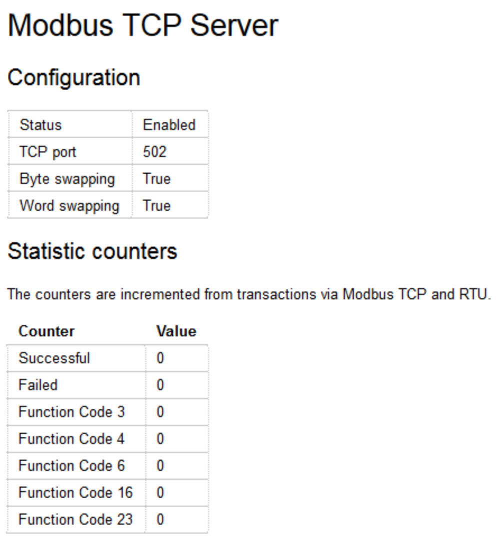
When the MicroFlex e190 or MotiFlex e180 drive is operating as a Modbus Server, Netdata events will only be raised whenever the Modbus Client writes *modified* data to Netinteger / Netfloat locations 0 to 31.

Diagnostics



The web page built into the MicroFlex e190 and MotiFlex e180 drive products contains diagnostic information about Modbus TCP and Modbus RTU Server (for e190) operation. Launch the web page from the Mint Sidebar and select ‘Menu”.

The Server diagnostics page displays the current configuration and counters showing the number of successful and unsuccessful Modbus TCP/RTU transactions (and also shows how many transactions of each Modbus function code type have been actioned).



Modbus TCP Server

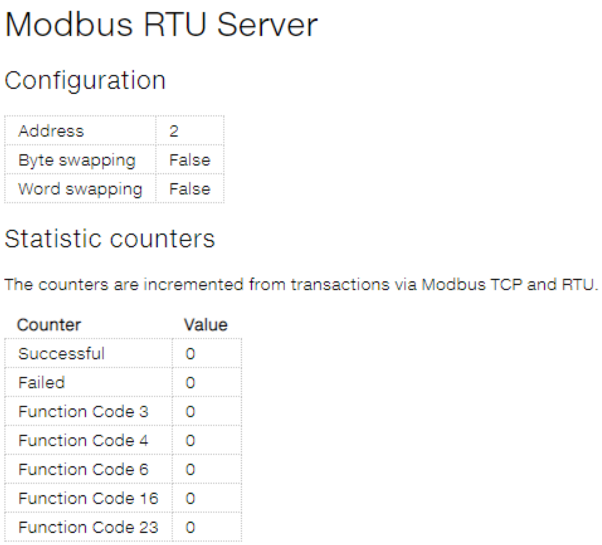
Configuration

Status	Enabled
TCP port	502
Byte swapping	True
Word swapping	True

Statistic counters

The counters are incremented from transactions via Modbus TCP and RTU.

Counter	Value
Successful	0
Failed	0
Function Code 3	0
Function Code 4	0
Function Code 6	0
Function Code 16	0
Function Code 23	0



Modbus RTU Server

Configuration

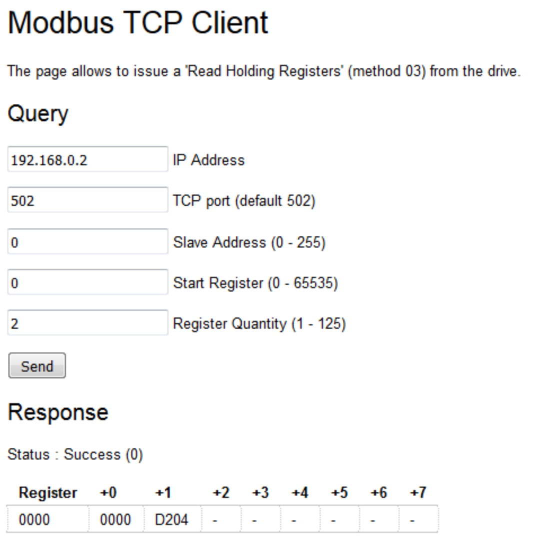
Address	2
Byte swapping	False
Word swapping	False

Statistic counters

The counters are incremented from transactions via Modbus TCP and RTU.

Counter	Value
Successful	0
Failed	0
Function Code 3	0
Function Code 4	0
Function Code 6	0
Function Code 16	0
Function Code 23	0

The Client diagnostics page lets the user initiate a ‘Read holding registers’ Modbus TCP function (function code 03). Diagnostics about the transaction are displayed after clicking the ‘Send’ button. This allows the user to test basic Modbus TCP operation without the need to write any Mint code…



Modbus TCP Client

The page allows to issue a 'Read Holding Registers' (method 03) from the drive.

Query

192.168.0.2 IP Address

502 TCP port (default 502)

0 Slave Address (0 - 255)

0 Start Register (0 - 65535)

2 Register Quantity (1 - 125)

Response

Status : Success (0)

Register	+0	+1	+2	+3	+4	+5	+6	+7
0000	0000	D204	-	-	-	-	-	-

NextMove e100 and ESB-2 Modbus configuration

These products are only capable of operating as Modbus Servers. Both products support Modbus RTU, only NextMove e100 supports Modbus TCP.
The Mint keyword ModbusParameter allows configuration of:

- Modbus enable/disable
- Register mapping (to comms or netdata locations)
- Byte order
- Word order
- Diagnostics

Configuration

All Modbus parameters are configured via a Mint keyword – ModbusParameter.
Before enabling Modbus operation it is necessary to set the correct byte and word order to suit the connected Modbus Client (Client) and to configure how Modbus registers in the received data packets are mapped to internal data areas in the NextMove controller.

value = ModbusParameter (bus, index)
ModbusParameter (bus, index) = value

Mint pre-defined constant values for bus are:

_busETHERNET	*5*	*For Modbus TCP parameters*
_busSERIAL1	*6*	*For Modbus RTU parameters*

Mint pre-defined constant values for index are:

_mpENABLE	*0*	*0 = Disabled, 1 = Enabled*
_mpREGISTER_MAPPING	*1*	*0 = NetData array (e100 default), 1 = Comms array (ESB-2 default)*
_mpBYTE_ORDER	*2*	*0 = Big Endian (default), 1 = Little Endian*
_mpWORD_ORDER	*3*	*0 = Big Endian (default), 1 = Little Endian*
_mpDROPPED_FRAMES	*6*	*Read only counter of number of invalid packets received*
_mpDEBUG	*7*	*Parameter used to display Modbus diagnostics*

Mint pre-defined constant values for _mpREGISTER_MAPPING are:

_rmNET_DATA	*0*	*To access netinteger / netfloat data*
_rmCOMMS_ARRAY	*1*	*To access commsinteger / comms data*

NextMove ESB-2 is only provided with Comms array data so attempting to configure a Netdata register mapping on this controller will result in a ‘Data Out of Range’ error.

The default byte and word orders are configured to match the requirements for communication with other ABB Motion, PLC and HMI products. Some third party Modbus products tend to use little endian word order so when using a third party Client it may be necessary to include…ModbusParameter (bus, _mpWORD_ORDER) = 1…in the Mint Startup block code.

The following baud rates are supported on all controllers:

- 19200
- 38400
- 57600
- 115200

Note that, due to timing constraints, 9600 baud is not supported. The baudrate to be used is set using the Mint SERIALBAUD(_Term1) keyword in the Mint program (or via the Mint Workbench 'Connectivity' page). As Modbus RTU packets include a node address it is also important to set the correct value for BUSNODE(_busSERIAL1) in the Mint program (or via the Mint Workbench 'Connectivity' page). Support is included for broadcast write functions (i.e. to Node address 0) so the user should avoid configuring a controller as Node 0 unless they intend to use broadcast functions.

When using Modbus TCP NextMove e100 uses the standard Modbus port 502. The IP address used by Modbus TCP is usually set by the rotary address switches on the front of the NextMove e100; 192.168.100.x (where x is the switch setting), but use of the BUSIPCONFIG keyword in the program will allow the controller to assume any IP address (please refer to the Mint Help file for further details). Similarly, if there is a MicroFlex e190 or MotiFlex e180 drive in an EPL network acting as a router (between standard Ethernet and Ethernet Powerlink) then the NextMove e100's IP address can appear to be on the standard Ethernet network – with the drive using a Network Address Translation (NAT) table to resolve this for example.

To enable Modbus Server (Server) operation the Mint program should issue….

ModbusParameter (bus, _mpENABLE) = 1 (where bus = _busETHERNET for Modbus TCP or _busSERIAL1 for Modbus RTU). Enabling Modbus RTU Server operation automatically disables both Host Comms Protocol (HCP1/2) and ABB Binary Protocol (BBP) functionality on the serial port. Enabling Modbus RTU also prevents the controller from directing data from Mint PRINT statements to the serial port to avoid corruption of Modbus data packets.

Example Mint Code

The following code snippets show typical Mint code that may be included in a NextMove controller's Startup block:

Example Mint code – Mint Modbus RTU Server (e.g. NextMove ESB-2) connected to AC500 or CP600 using Comms array:

```
BUSNODE(_busSERIAL1) = 2        'Mint controller is node 2 on RTU network
SERIALBAUD(_Term1) = 57600      'Running at 57.6kbaud
ModbusParameter (_busSERIAL1, _mpBYTE_ORDER)   = 0  'Use big endian byte order
ModbusParameter (_busSERIAL1, _mpWORD_ORDER) = 0  'Use big endian word order
ModbusParameter (_busSERIAL1, _mpREGISTER_MAPPING) = _rmCOMMS_ARRAY
ModbusParameter (_busSERIAL1, _mpENABLE) = 1
```

Example Mint code – Mint Modbus TCP Server (e.g. NextMove e100) connected to AC500 or CP600 using Netdata array:

```
ModbusParameter (_busETHERNET, _mpBYTE_ORDER)   = 0  'Use big endian byte order
ModbusParameter (_busETHERNET, _mpWORD_ORDER) = 0  'Use big endian word order
ModbusParameter (_busETHERNET, _mpREGISTER_MAPPING) = _rmNET_DATA
ModbusParameter (_busETHERNET, _mpENABLE) = 1
```

Example Mint code – Mint Modbus RTU Server (e.g. NextMove ESB-2) connected to third party Modbus Client using little endian word order and Netdata:

```
BUSNODE(_busSERIAL1) = 4        'Mint controller is node 4 on RTU network
SERIALBAUD(_Term1) = 38400      'Running at 38.4kbaud
ModbusParameter (_busSERIAL1, _mpBYTE_ORDER)   = 0  'Use big endian byte order
ModbusParameter (_busSERIAL1, _mpWORD_ORDER) = 1  'Use little endian word order
ModbusParameter (_busSERIAL1, _mpREGISTER_MAPPING) = _rmNET_DATA
ModbusParameter (_busSERIAL1, _mpENABLE) = 1
```

As NextMove e100 supports both Ethernet and serial channels it is possible to configure / enable Modbus TCP operation on Ethernet and at the same time configure / enable Modbus RTU operation on the serial port.

Mint events

Mint events operate differently depending on the controller being used:
NextMove ESB-2: Comms events will be raised whenever the Modbus Client writes to Comms locations 1 to 5 (i.e. data does not necessarily have to change for an event to be raised).

NextMove e100: Comms events will be raised whenever the Modbus Client writes *modified* data to Comms locations 1 to 10 (i.e. data must change for an event to be raised). Netdata events operate in the same manner and will only be raised whenever the Modbus Client writes modified data to Netinteger / Netfloat locations 0 to 31.

Diagnostics

The ModbusParameter keyword provides two types of diagnostics.

1. Reading ModbusParameter (bus, _mpDROPPED_FRAMES) will return a counter indicating how many Modbus packets have been rejected (e.g. invalid checksum, incomplete message……in summary, any message for which neither a valid nor exception response has been generated)
2. Writing a terminal channel value to ModbusParameter (bus, _mpDEBUG) will reset diagnostic counters and setup which terminal channel is to be used to display diagnostic information. Reading ModbusParameter (bus, _mpDEBUG) will then return which terminal channel is being used for diagnostics and will display some summary information on the selected terminal channel (e.g. last packet received, last packet transmitted, error counters etc..)

Example:

```
ModbusParameter (_busETHERNET, _mpDEBUG) = _Term2
? ModbusParameter (_busETHERNET, _mpDEBUG)
```

If these commands were entered on a NextMove e100 the controller would display diagnostic information about Modbus TCP on the USB terminal (i.e. Workbench terminal window).

Example debug output:

```
Valid: 2
Dropped: 0.  Exception: NO_EXCEPTION
Rx: 8 bytes
Node: 2   Fn code (hex): 03
Data (hex): 00 02 00 01
CRC (hex): f9 25
Tx: 7 bytes
Node: 2   Fn code (hex): 03
Data (hex): 02 00 00
CRC (hex): 44 fc
```

Contact us

For more information please contact your
local ABB representative or one of the following:

new.abb.com/motion
new.abb.com/drives
new.abb.com/drivespartners
new.abb.com/PLC