Using Modbus TCP with Brainboxes products

Contents

This manual applies to the following Ethernet I/O products:

Brainboxes Limited, 18 Hurricane Drive, Liverpool International Business Park, Speke, Liverpool, Merseyside, L24 8RL
Tel: +44 (0)151 220 2500 Fax: +44 (0)151 252 0446
www.brainboxes.com | sales@brainboxes.com | support@brainboxes.com

1 Scope of this document

This document describes the implementation of the Modbus TCP protocol with the Brainboxes Ethernet I/O (ED) product range. The document will explain what the protocol is, how it works and how to use it with Brainboxes ED range.

2 What is Modbus TCP?

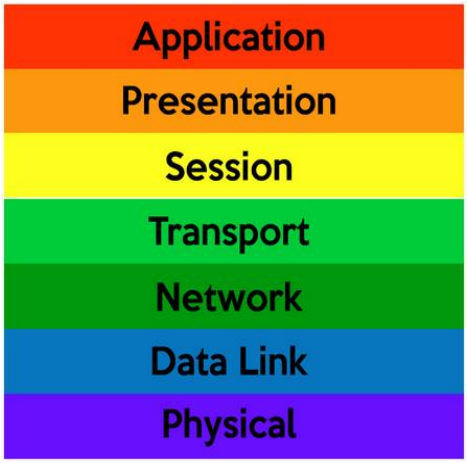
Modbus Protocol is a messaging structure developed by Modicon in 1979. It is used to establish master-slave/client-server communication between intelligent devices. It is a de facto standard, truly open and the most widely used network protocol in the industrial manufacturing environment. It has been implemented by hundreds of vendors on thousands of different devices to transfer discrete/analog I/O and register data between control devices.

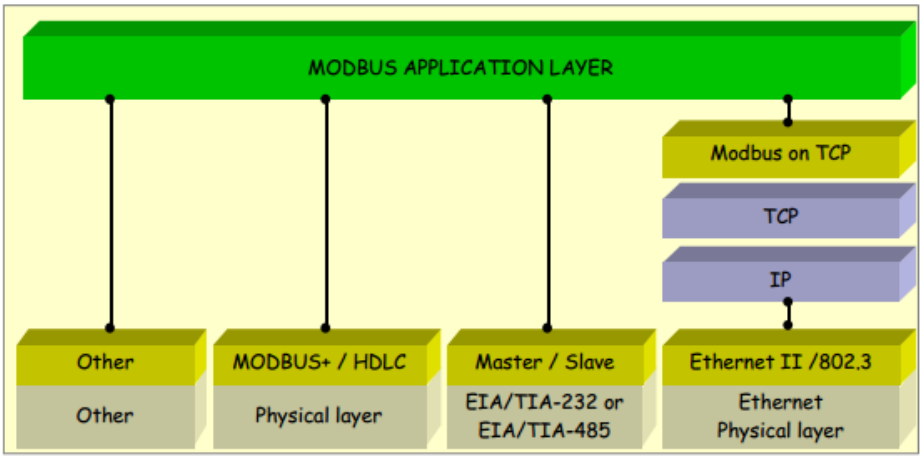
TCP/IP is the most common transport protocol used over the Internet which is actually a set of layered protocols, providing a reliable data transport tool between machines.

Combining a widespread physical network (Ethernet) with a universal networking standard (TCP/IP) and a vendor-neutral data representation, Modbus gives a truly open, accessible network for exchange of process data.

Modbus TCP/IP has become so popular due to its openness, simplicity, low-cost development, and minimum hardware required to support it.

The Modbus protocol is located within the seventh layer of the OSI model, the application layer. Modbus allows client/server communication between devices connected on different types of buses and networks.





Modbus is a request/reply protocol and offers services specified by function codes.
Modbus function codes are elements of Modbus request/reply PDUs.

There are currently three implementations ofModbus:

- TCP/IP over Ethernet.
- Asynchronous serial transmission over a variety of media (wire: EIA/TIA-232-E, EIA-422, EIA/TIA-485-A; fiber, radio, etc.)**Not implemented in our products.**
- Modbus PLUS, a high speed token passing network.**Not implemented in our products.**

A communicating system over Modbus TCP/IP may include different types of device:

- A Modbus TCP/IP Client and Server devices connected to a TCP/IP network
- The Interconnection devices like bridge, router or gateway for interconnection between the TCP/IP network and a serial line sub-network which permit connections of Modbus Serial line Client and Server end devices.

3 Modbus TCP settings on Brainboxes webpage

Brainboxes ED products by default are configured to use the ASCII protocol. To enable the Modbus protocol instead, the user must access the webpage for their device. The webpage for our Ethernet I/O products can be accessed by simply entering the IP address of the device in to the URL bar in your web browser. Here is an example:

If you wish to directly access the protocol page straight away so that you can configure the Modbus settings of your device quickly, you can enter the following in to the URL bar:
192.168.0.62/#/protocol

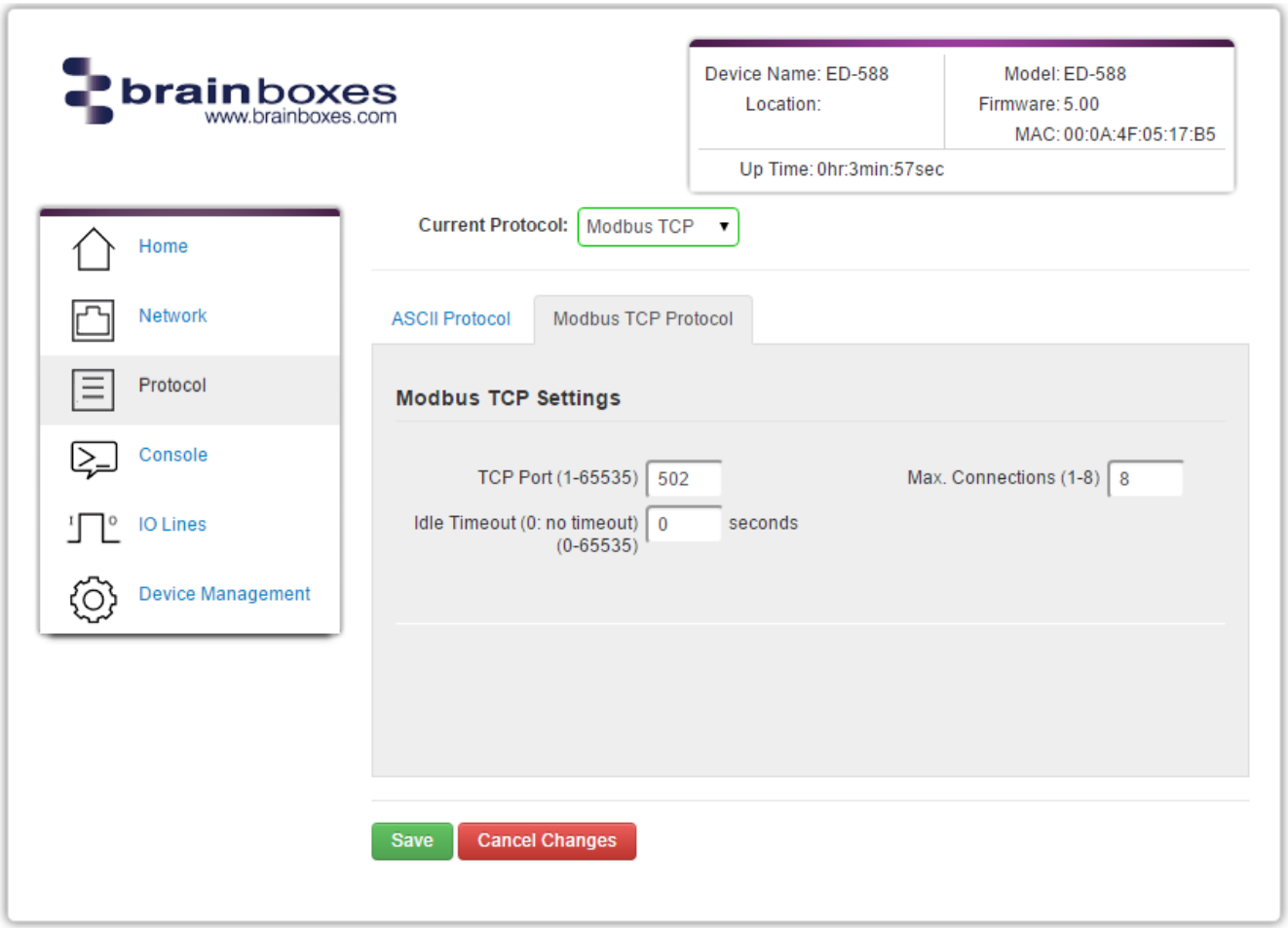
Note: The IP address listed here is purely an example. The user will need to enter the IP address of their device, this may have been set automatically by a DHCP server on the users network, be the factory default address of 192.168.127.254 or may been previously set the user via the device webpage to a preferred value.

Once the user has accessed the Modbus protocol page for their device, they will have the option to configure:

- TCP Port (default setting = 502)
- Idle Timeout value (default setting = 0)
- Number of Max Connections (default setting = 8)

One last thing which the user must ensure is that the **Current Protocol** drop down tab is set to Modbus TCP and not ASCII mode.

The image below is an example of the Modbus protocol page for an ED-588:



The **TCP port**is a 16 bit number, 1 – 65535, used to identify the services or processes being used in networking communications. Specific port numbers are often used to identify specific services. By convention, TCP port 502 is used by the Modbus protocol.

When the **Idle Timeout** is set, if there is no communication to the device for the specified period of time (in seconds), the connection will be closed. The default idle connection is 0, meaning the connection will never be dropped automatically.

The **Max Connections** field allows the user to select the maximum amount of simultaneous connections which can be made to their ED device at any one time. This field accepts any values between 1 (minimum) and 8 (maximum).

4 Logical addressing

Within the messages passed between Modbus devices, the addresses for registers, coils and inputs are always two-byte values, which can express values from 0 to 65535. Each of the types of addressable object (coils, discrete inputs, holding registers and input registers) have their own independent "address space": there can be a coil with address 123 and a holding register with address 123, and there is no relationship implied between them. So when using logical addressing, the type of object/access type always has to be stated as well. A logical address may be written as a decimal or a hexadecimal number: we write them as hexadecimal, indicated by a '0x' prefix.

5 984 style addressing

Modbus started life as a proprietary standard based on a family of programmable controllers, and the addressing notation from the early version of this standard, although now officially superseded, is still widely used. It is often called 984 addressing, after the model of programmable controller which popularised it. In this address notation, the address is always written in decimal, with an offset of 1 from the logical address. It is then padded out with leading zeroes to 4 digits (so logical address 20 becomes 0021), and then a prefix digit is added to indicate which address space is to be used, making a 5-digit written address. So, logical address 20 would become 00021 if it referred to a coil address, 10021 if it was a discrete input, 30021 for an input register or 40021 for a holding register. You may also see 4-digit or 6-digit versions of this scheme.

984 addresses	Type	Logical addresses
00001-09999	Coil	0-9998 (decimal) 0x0000-0x270E (hexadecimal)
10001-19999	Discrete input	0-9998 (decimal) 0x0000-0x270E (hexadecimal)
30001-39999	Input register	0-9998 (decimal) 0x0000-0x270E (hexadecimal)
40001-49999	Holding register	0-9998 (decimal) 0x0000-0x270E (hexadecimal)

The prefix digit is sometimes used as shorthand for the type of access: '0x' referring to coils (not to be confused with the 0x which indicates a hexadecimal number!),'1x' referring to discrete inputs, '3x' referring to holding registers, and '4x' referring to input registers.

6 IEC 61131 addressing

Programmable controllers and HMIs often use the IEC 61131 standard for referring to internal 1-bit values (%M0, %M1, …) and 16-bit values (%MW0, %MW1, …). This notation is sometimes also applied to Modbus addressing, with the %M0, %M1, … addresses referring to Modbus coils, andthe %MW0, %MW1, … addresses referring to Modbus holding registers. There is no way in this scheme to represent the read-only types.

IEC 61131 addresses	Type	Logical addresses
%M0-%M65535	Coil	0-65535 (decimal) 0x0000-0xFFFF (hexadecimal)
N/a	Discrete input	Not accessible in this format
N/a	Input register	Not accessible in this format
%MW0 - %MW65535	Holding register	0-65535 (decimal) 0x0000-0xFFFF (hexadecimal)

7 Modbus 1.1b3 standard addressing

If you download the latest version (1.1b3) of the Modbus standard, you will find that it uses yet another addressing style. In what it calls the "Modbus data model", the address of each object starts at 1, i.e. it is the logical address plus 1. You can see in the examples that the object addresses are always 1 greater than the values actually transferred in Modbus data packets. Like the logical address, an address written this way does not specify what is being addressed; the type of object/access needs to be stated as well.

Modbus data model addresses	Type	Logical addresses
1-65536	Coil (1 bit)	0-65535 (decimal) 0x0000-0xFFFF (hexadecimal)
1-65536	Discrete input (1 bit)	0-65535 (decimal) 0x0000-0xFFFF (hexadecimal)
1-65536	Input register (16 bit)	0-65535 (decimal) 0x0000-0xFFFF (hexadecimal)
1-65536	Holding register (16 bit)	0-65535 (decimal) 0x0000-0xFFFF (hexadecimal)

8 Modbus data formats

Modbus uses the concept of a *data table* to refer to data. A data table is an array or block of memory used to store data. Data is referenced using data table addresses. Modbus data table addresses come in four types:

- **Discrete inputs**–Represent a single bit (Boolean) which can only be read. In other words, the client can only perform a read action on the discrete inputs.
- **Coils** - These are read-write Boolean values. They are typically used to represent outputs or internal bits which are both read by and written to by the user.
- **Input registers** - These are read only 16 bit integers. They are typically used to represent analogue input values and other integer values which are read but not written to by the user.
- **Holding registers** - These are read-write 16 bit integers. They are typically used to represent analogue outputs or internal numbers which are both read by and written to by the user.

9 Product data tables

9.1 ED-588

	Modbus access type	Modbus function codes	Logical address	984 style address	IEC 61131 address
Read digital inputs	Coil	1	0x0020 – 7	00033 – 40	%M32 – 39
Read digital inputs	Discrete input	2	0x0000 – 7	10001 – 8	N/A
Read digital inputs	Input register	4	0x0020	30033	N/A
Read DI counter values	Input register	4	0x0000 – 7	30001 – 8	N/A
Read DI counter values	Holding register	3	0x0000 – 7	40001 – 8	%MW0 – 7
Clear DI counters	Coil	5, 15	0x0200 – 7	00513 – 20	%M512 – 19
Set/read digital outputs	Coil	1, 5, 15	0x0000 – 7	40033	%M0 - 7
Set/read digital outputs	Holding register	3, 6, 16	0x0020	40033	%MW32
Output overload flags	Discrete input	2	0x0400 – 7	11025 – 32	N/A
Output overload flags	Input register	4	0x0400	31025	N/A

9.2 ED-516

	Modbus access type	Modbus function codes	Logical address	984 style address	IEC 61131 address
Read digital inputs	Coil	1	0x0020 – F	00033 – 48	%M32 – 37
Read digital inputs	Discrete input	2	0x0000 – F	10001 – 16	N/A
Read digital inputs	Input register	4	0x0020	30033	N/A
Read DI counter values	Input register	4	0x0000 – F	30001 – 16	N/A
Read DI counter values	Holding register	3	0x0000 – F	40001 – 16	%MW0 – 15
Clear DI counters	Coil	5, 15	0x0200 – F	00513 – 28	%M512 – 27

9.3 ED-538

	Modbus access type	Modbus function codes	Logical address	984 style address	IEC 61131 address
Read digital inputs	Coil	1	0x0020 – 7	00033 – 40	%M32 – 39
Read digital inputs	Discrete input	2	0x0000 – 7	10001 – 8	N/A
Read digital inputs	Input register	4	0x0020	30033	N/A
Read DI counter values	Input register	4	0x0000 – 7	30001 – 8	N/A
Read DI counter values	Holding register	3	0x0000 – 7	40001 – 8	%MW0 – 7
Clear DI counters	Coil	5, 15	0x0200 – 7	00513 – 20	%M512 – 19
Set/read digital outputs	Coil	1, 5, 15	0x0000 – 3	00001 – 4	%M0 – 3
Set/read digital outputs	Holding register	3, 6, 16	0x0020	40033	%MW32
Output overload flags	Discrete input	2	0x0400 – 3	11025 – 8	N/A
Output overload flags	Input register	4	0x0400	31025	N/A

9.4 ED-527

	Modbus access type	Modbus function codes	Logical address	984 style address	IEC 61131 address
Set/read digital outputs	Coil	1, 5, 15	0x0000 – F	00001 – 16	%M0 – 15
Set/read digital outputs	Holding register	3, 6, 16	0x0020	40033	%MW32
Output overload flags	Discrete input	2	0x0400 – F	11025 – 40	N/A
Output overload flags	Input register	4	0x0400	31025	N/A

10 Worked example

10.1 Writing a single coil

In this example we are sending Modbus commands to write single bits to an ED-527 which are either high or low.

To write a single bit as either high or low, we use the Write Single Coil function code: 05

The data value of 0x0000 to output the bit as low.

The data value of 0xFF00 to output the bit as high.

The ED-527 has its 16 output register addresses starting at 0x0001 through to 0x0010.

To set the bit low in register 1, the Modbus packet will look like this:

0017 0000 0006 FF 05 0001 FF00

This packet is broken down in to the following fields:

TrID	Prot	Len	UI	Fn	Addr	Data
0017	0000	0006	FF	05	0001	FF00

- TrID: Transaction ID.
- Prot: Is the protocol which is always 00 for Modbus TCP.
- Len: Is the number of bytes in the rest of the transaction.
- Fn: Is the Modbus function code for writing a single coil: 05
- Addr: Is the address of the Modbus register of which we are setting the I/O state.
- Data: Is the data we are writing. 0x0000 sets the bit low and 0xFF00 sets the bit high.

10.2 Reading a one bit value

There is no Modbus command to read back just a single output bit, instead you just use the Read Multiple Coils Function Code: x01, and set the number of coils to be read to 1.

The ED-527 has its 16 output register address starting at 0x0000 through to 0x000F.

To read the value of the bit in register 1, the Modbus data packet looks like this:

TrID	Prot	Len	UI	Fn	Addr	Data
0065	0000	0006	FF	01	0000	0001

- TrID: is the transaction ID (start with 1 and increment for each transfer you do)
- Prot: is the protocol which is always 00 for Modbus TCP
- Len: is the number of bytes in the rest of the transaction
- UI: is the Unit Identifier =0xFF
- Fn: is the Modbus Function code 01= Read Multiple Coils
- Addr: Is the address of the Modbus register from which the first bit read is taken
- Data: is the number of registers bits we are intending to read, =01 here we are reading just 1 bit value

The ED device responds with:

TrID	Prot	Len	UI	Fn	BC	Data
0065	0000	0004	FF	01	01	01

- TrID: Is the same transaction ID the PC used in its request
- Prot: Is the protocol which is always 00 for Modbus TCP
- Len: Is the number of bytes in the rest of the transaction
- UI: Is the Unit Identifier: 0xFF
- Fn: Is the Modbus Function code: 01 - Read Multiple Coils
- BC: Is the byte count of data to follow
- Data: Is the value of the bit in the register

10.3 Reading a 16 bit value

To read all 16 ED-527 output bits back use the Read Multiple Coils Function, code: x01, and set the number of coils to be read to 16: 0x10.

The ED-527's 16 output register addresses start at 0x0001 through to 0x0010.

To read the value of the bit in register 1 the Modbus data packet looks like this:

TrID	Prot	Len	UI	Fn	Addr	Data
0066	0000	0006	FF	01	0000	0010

- TrID: is the transaction ID (start with 1 and increment for each transfer you do)
- Prot: is the protocol which is always 00 for Modbus TCP
- Len: is the number of bytes in the rest of the transaction
- UI: is the Unit Identifier =0xFF
- Fn: is the Modbus Function code 01= Read Multiple Coils
- Addr: Is the address of the Modbus register from which the first bit read is taken. Note if we try and read 16 registers from any address other than 0x0000 the ED-527 will report an error as it has a maximum of 16 registers.
- Data: is the number of registers bits we are intending to read, =ox10 here we are reading 16 x 1 bit registers

The ED device responds with:

TrID	Prot	Len	UI	Fn	BC	Data
0066	0000	0005	FF	01	02	2B00

- TrID: is the same transaction ID the PC used in its request
- Prot: is the protocol which is always 00 for Modbus TCP
- Len: is the number of bytes in the rest of the transaction
- UI: is the Unit Identifier =0xFF
- Fn: is the Modbus Function code 01= Read Multiple Coils
- BC: Is the byte count of data to follow in this case 2 bytes
- Data: is the value of the bit in the 16 registers the firs byte 2B is the lower register and the last 0x00 is the higher byte register

10.4 Writing single bit

We can use Modbus commands to write single bits of the ED-527 either high or low.

To write a single bit either high or low we use the Write Single Coil Function, code: 0x05.

The data value is 0x0000 to output the bit low and the data value is 0xFF00 to output the bit high.

The ED-527 has its 16 single bit output registers address starting at 0x0000 through to 0x000F.

To set the bit low in register 0 the Modbus data packet looks like this:

TrID	Prot	Len	UI	Fn	Addr	Data
0017	0000	0006	FF	05	0000	0000

To set the bit high in register 0 the Modbus data packet looks like this:

TrID	Prot	Len	UI	Fn	Addr	Data
0017	0000	0006	FF	05	0000	FF00

- Prot: is the protocol which is always 00 for Modbus TCP
- Len: is the number of bytes in the rest of the transaction
- UI: is the Unit Identifier =0xFF
- Fn: is the Modbus Function code 05= Write Single Coil
- Addr: Is the address of the Modbus register of which we are setting the I/O state
- Data: is the data we are writing, 0x0000 sets the bit Low, 0xFF00 sets the bit High

The ED responds by echoing the data back to the master i.e. PC or PLC.

10.5 Writing 16 bits

We can use Modbus commands to write multiple bits to the ED-527, either high or low using the Write Multiple Coil Function, code: 0x0F.

The ED-527 has 16 single bit output registers which addresses start at 0x0000 through to 0x000F.

We send 0xEE to the lower 8 bits of the ED-527 output and 0xAA to the upper bits of the ED-527 outputs.

To set the 16 bits starting at register 0 the Modbus data packet looks like this:

TrID	Prot	Len	UI	Fn	Addr	#Bits	BC	Data
0018	0000	0006	FF	0F	0000	0010	02	EEAA

- TrID: is the transaction ID
- Prot: is the protocol which is always 00 for Modbus TCP
- Len: is the number of bytes in the rest of the transaction
- UI: is the Unit Identifier: 0xFF
- Fn: is the Modbus Function, code: 0F
- Addr: Is the address of the first Modbus register of which we are setting the I/O state
- #Bits is the count of the number of Bits we are writing
- BC is the byte count of the number of bytes following
- Data: is the data we are writing, 0xEEAA in this example

The ED responds with:-

TrID	Prot	Len	UI	Fn	Addr	#Bits
0018	0000	0006	FF	0F	0000	0010