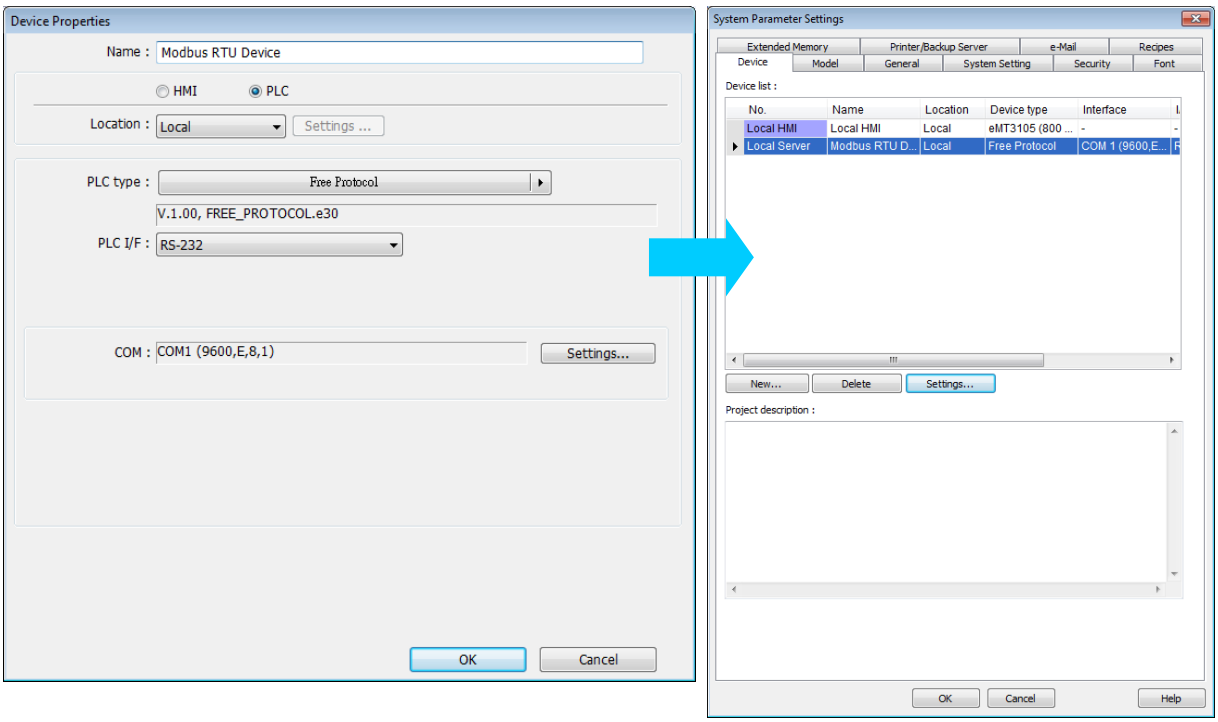
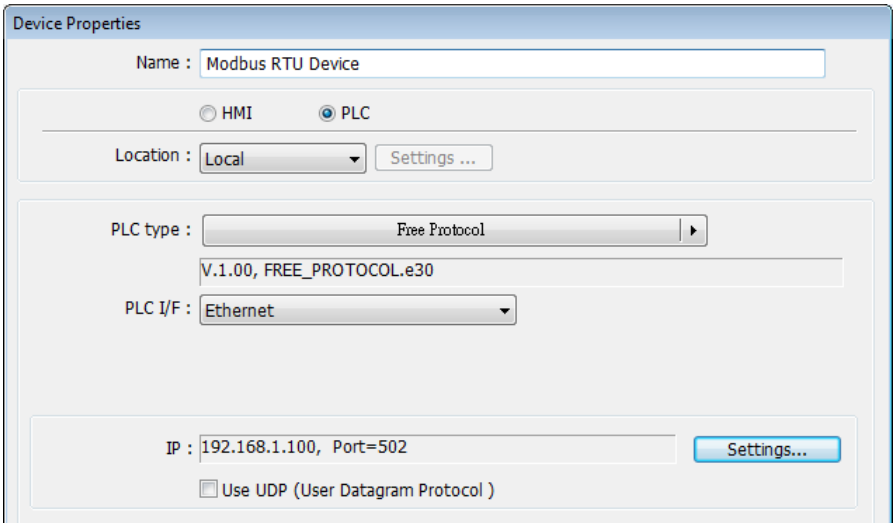
18.11. Use the Free Protocol to Control a Device

If EasyBuilder Pro does not provide a driver for a specific device, users can use OUTPORT and INPORT built-in functions to control the device. The data sent by OUTPORT and INPORT must follow the communication protocol of the device. The following example explains how to use these two functions to control a MODBUS RTU device.

1. First, create a new device in the device table. The device type of the new device is set to "Free Protocol" and named with "MODBUS RTU device" as follows:



2. The interface of the device (PLC I/F) uses [RS-232]. If a MODBUS TCP/IP device is connected, the interface should be [Ethernet] with correct IP and port number as follows:



Suppose that the HMI will read the data of 4x_1 and 4x_2 on the device. First, utilize OUTPORT to send out a read request to the device. The format of OUTPORT is:

OUTPORT(command[start], device_name, cmd_count)

Since “MODBUS RTU device” is a MODBUS RTU device, the read request must follow MODBUS RTU protocol. The request uses”Reading Holding Registers (0x03)” command to read data. The following picture displays the content of the command. (The items of the station number (byte 0) and the last two bytes (CRC) are ignored).

Request

Function code	1 Byte	0x03
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	0x03
Byte count	1 Byte	2 x N*
Register value	N* x 2 Bytes	

*N = Quantity of Registers

Error

Error code	1 Byte	0x83
Exception code	1 Byte	01 or 02 or 03 or 04

Depending on the protocol, the content of a read command as follows (The total is 8 bytes):

command[0]: station number (BYTE 0)
command[1]: function code (BYTE 1)
command[2]: high byte of starting address (BYTE 2)
command[3]: low byte of starting address (BYTE 3)
command[4]: high byte of quantity of registers (BYTE 4)
command[5]: low byte of quantity of registers (BYTE 5)
command[6]: low byte of 16-bit CRC (BYTE 6)
command[7]: high byte of 16-bit CRC (BYTE 7)

So a read request is designed as follows:

```
char command[32]
short address, checksum

FILL(command[0], 0, 32)    //    initialize command[0]~command[31] to 0

command[0] = 0x1    //    station number
command[1] = 0x3    //    read holding registers (function code    is 0x3)

address =    //    starting address (4x_1) is 0
HIBYTE(address, command[2])
LOBYTE(address, command[3])

read_no = 2    //    the total words of rading is 2 words
HIBYTE(read_no, command[4])
LOBYTE(read_no, command[5])

CRC(command[0], checksum, 6)    //    calculate 16-bit CRC

LOBYTE(checksum, command[6])
HIBYTE(checksum, command[7])
```

Lastly, use OUPORT to send out this read request to PLC.

```
OUTPORT(command[0], “MODBUS RTU Device”, 8)    //    send read request
```

After sending out the request, use INPORT to get the response from PLC. Depending on the protocol, the content of the response is as follows (the total byte is 9):

command[0]: station number (BYTE 0)
command[1]: function code (BYTE 1)
command[2]: byte count (BYTE 2)
command[3]: high byte of 4x_1 (BYTE 3)
command[4]: low byte of 4x_1 (BYTE 4)
command[5]: high byte of 4x_2 (BYTE 5)
command[6]: high byte of 4x_2 (BYTE 6)
command[7]: low byte of 16-bit CRC (BYTE 7)
command[8]: high byte of 16-bit CRC (BYTE 8)

The format of INPORT is:

```
INPORT(response[0], “MODBUS RTU Device”, 9, return_value)    //    read reponse
```

Where the real read count is restored to the variable return_value (unit is byte). If return_value is 0, it means reading fails in executing INPORT.

According to the MODBUS RTU protocol specification, the correct response[1] must be equal to 0x03. After getting correct response, calculate the data of 4x_1 and 4x_2 and put in the data into LW-100 and LW-101 of HMI.

```
If (return_value) >0 and response[1] == 0x3) then
   read_data[0] = response[4] + (response[3] << 8) // 4x_1
   read_data[1] = response[6] + (response[5] << 8) // 4x_2

   SetData(read_data[0], “Local HMI”, LW, 100, 2)
endif
```

The complete macro is as follows:

```
// Read Holding Registers
macro_command main()

   char command[32], response[32]
   short address, checksum
   short read_no, return_value, read_data[2], i

   FILL(command[0], 0, 32)//    initialize command[0]~command[31] to 0
   FILL(response[0], 0, 32)

   command[0] = 0x1//    station number
   command[1] = 0x3//    read holding registers (function code is 0x3)

   address = 0
   address =    0//    starting address (4x_1) is 0
   HIBYTE(address, command[2])
   LOBYTE(address, command[3])

   read_no = 2/    the total words of reading is 2 words
   HIBYTE(read_no, command[4])
   LOBYTE(read_no, command[5])

   CRC(command[0], checksum, 6)//    calculate 16-bit CRC

   LOBYTE(checksum, command[6])
   HIBYTE(checksum, command[7])

   OUTPORT(command[0], "MODBUS RTU Device", 8 )//    send request
   INPORT(response[0], "MODBUS RTU Device", 9, return_value)//    read response

   if (return_value > 0 and response[1] == 0x3) then
      read_data[0] = response[4] + (response[3] << 8)//    4x_1
      read_data[1] = response[6] + (response[5] << 8)//    4x_2

      SetData(read_data[0], "Local HMI", LW, 100, 2)
   end if

   end macro_command
```

The following example explains how to design a request to set the status of 0x_1. The request uses ”Write Single Coil(0x5)” command.

Request

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Response

Function code	1 Byte	**0x05**
Output Address	2 Bytes	0x0000 to 0xFFFF
Output Value	2 Bytes	0x0000 or 0xFF00

Error

Error code	1 Byte	**0x85**
Exception code	1 Byte	01 or 02 or 03 or 04

The complete macro is as follows:

```
//    Write Single Coil (ON)
macro_command main()

char command[32], response[32]
short address, checksum
short i, return_value

FILL(command[0], 0, 32)//    initialize command[0]~ command[31] to 0
FILL(response[0], 0, 32)

command[0] = 0x1//    station number
command[1] = 0x5//    function code : write single coil

address = 0
HIBYTE(address, command[2])
LOBYTE(address, command[3])

command[4] = 0xff//    force 0x_1 on
command[5] = 0

CRC(command[0], checksum, 6)

LOBYTE(checksum, command[6])
HIBYTE(checksum, command[7])

OUTPORT(command[0], "MODBUS RTU Device", 8)//    send request
INPORT(response[0], "MODBUS RTU Device", 8, return_value)//    read response

end macro_command
```