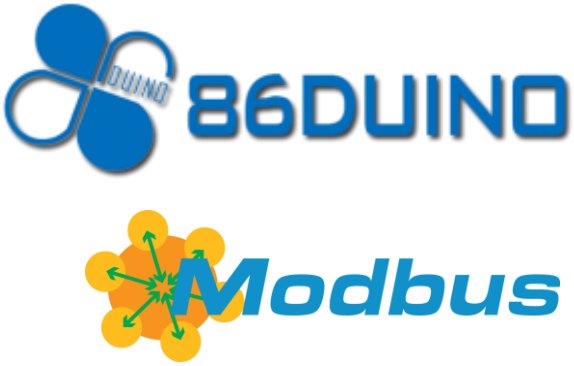

User Manual

Modbus API

86Duino Coding IDE 500+

Modbus Library

(Version1.1)

REVISION

DATE	VERSION	DESCRIPTION
2025/1/24	Version1.0	New Release.
2025/2/6	Version1.1	1. Fix repeated words “Register”. 2. Revise the return value and description for API functions with exception code.

COPYRIGHT

Ver.1.1 February, 2025

TRADEMARKS ACKNOWLEDGMENT

For more detailed information or if you are interested in other ICOP products, please visit our official websites at:

- Global: www.icop.com.tw
- USA: www.icoptech.com
- Japan: www.icop.co.jp
- Europe: www.icoptech.eu
- China: www.icop.com.cn

For technical support or drivers download, please visit our websites at:

- https://www.icop.com.tw/resource_entrance

For EtherCAT solution service, support or tutorials, 86Duino Coding IDE 500+ introduction, functions, languages, libraries, etc. Please visit the QEC website:

- QEC: https://www.qec.tw/

This Manual is for the QEC series.

SAFETY INFORMATION

- Read these safety instructions carefully.
- Please carry the unit with both hands and handle it with caution.
- Power Input voltage +19 to +50VDC Power Input (Typ. +24VDC)
- Make sure the voltage of the power source is appropriate before connecting the equipment to the power outlet.
- To prevent the QEC device from shock or fire hazards, please keep it dry and away from water and humidity.
- Operating temperature between -20 to +70°C/-40 to +85°C (Option).
- When using external storage as the main operating system storage, ensure the device's power is off before connecting and removing it.
- Never touch un-insulated terminals or wire unless your power adaptor is disconnected.
- Locate your QEC device as close as possible to the socket outline for easy access and avoid force caused by the entangling of your arms with surrounding cables from the QEC device.
- If your QEC device will not be used for a period of time, make sure it is disconnected from the power source to avoid transient overvoltage damage.

WARNING!

DO NOT ATTEMPT TO OPEN OR TO DISASSEMBLE THE CHASSIS (ENCASING) OF THIS PRODUCT. PLEASE CONTACT YOUR DEALER FOR SERVICING FROM QUALIFIED TECHNICIAN.

Content

Ch. 1

Introduction

1.1 About Modbus Library

86Diuno IDE includes this library from Coding version 318 to support the Modbus communication protocol, an industrial communication standard published in 1979 for communication between automated electronic devices such as Programmable logic controllers (PLCs).

86Duino Modbus library for communicating with Modbus over RS232/485/Ethernet (via RTU/ASCII/TCP protocol).

Modbus is a master/slave based protocol where a master node communicates with multiple slave nodes on the network. Each node has a unique address. When the master node sends a packet to the specified address, only the slave node at the corresponding address receives and parses the packet and executes and responds to commands based on the packet contents.

Modbus allows multiple devices (approximately 240) to connect and communicate on the same network. For example, a device that measures temperature and humidity can transmit the results to a computer. In Supervisory Control and Data Acquisition (SCADA) systems, Modbus is commonly used to connect monitoring computers with Remote Terminal Units (RTUs).

86Duino's Modbus library has the following features:

- It supports Modbus RTU, TCP, and ASCII sub-protocols.
- Runs as both Modbus master and Modbus slave nodes.
- Support Modbus gateway function

1.1.1 What is 86Duino IDE?

The 86Duino integrated development environment (IDE) software makes it easy to write and upload code to 86Duino boards and QEC MDevice. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software, which can be downloaded from https://www.qec.tw/software/.

For more information, please visit 86Duino Website.

1.2 Protocol Versions

The Modbus protocol currently exists in versions designed for serial ports, Ethernet, and other networks that support internet protocols.

Most Modbus device communications are conducted through the EIA-485 physical layer.

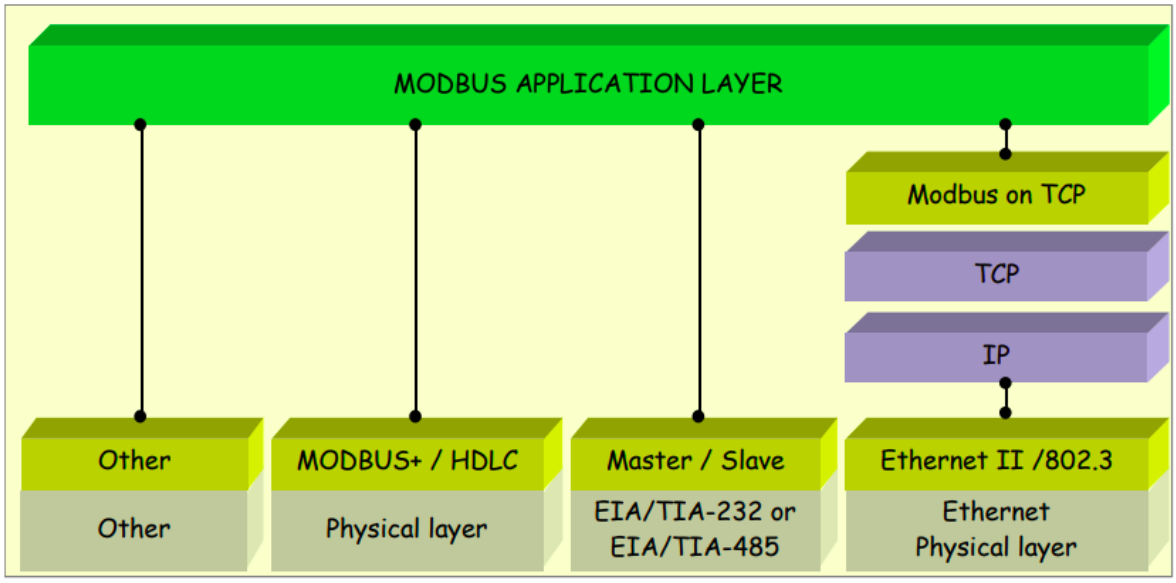

Image Source: Wikipedia (https://en.wikipedia.org/wiki/Modbus)

For serial connections, there are two variants, which differ slightly in numerical data representation and protocol details. Modbus RTU is a compact format that represents data in binary, while Modbus ASCII is a human-readable, more verbose format. Both variants use serial communication. The RTU format appends a Cyclic Redundancy Check (CRC) checksum to subsequent commands/data, whereas the ASCII format uses a Longitudinal Redundancy Check (LRC) checksum. Nodes configured in the RTU variant will not communicate with nodes set to the ASCII variant, and vice versa.

For connections via TCP/IP (such as Ethernet), there are multiple Modbus/TCP variants that do not require checksum calculations.

For all three communication protocols, the data model and function calls are the same, differing only in the method of encapsulation.

1.3 Communication and Devices

The Modbus protocol is a master/slave architecture. There is one master node, while other nodes participating in communication using the Modbus protocol are slave nodes. Each slave device has a unique address. In serial and Modbus Plus (MB+) networks, only the node designated as the master node can initiate a command. On Ethernet networks, any device can send a Modbus command, but typically, only one master device initiates commands.

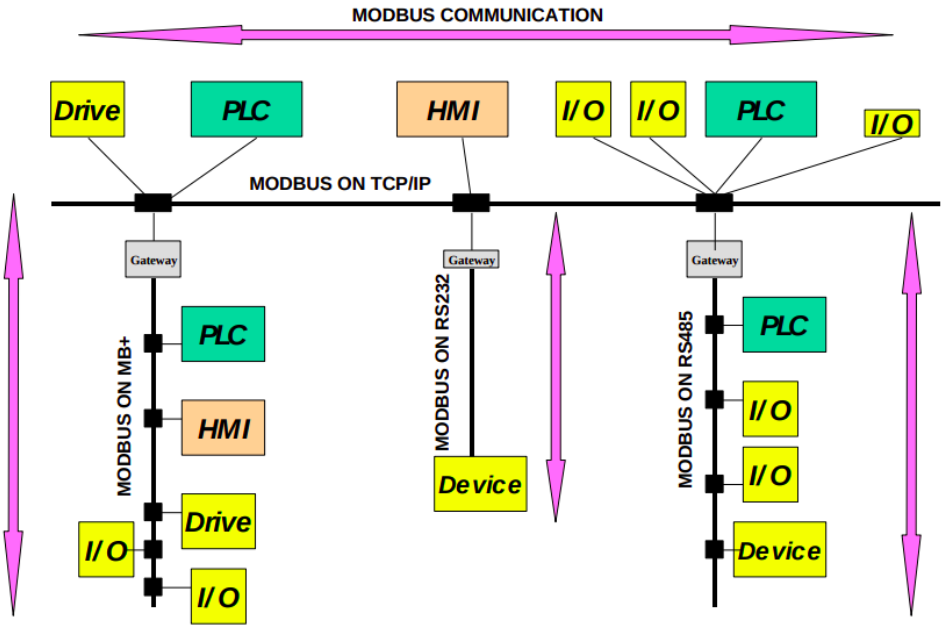

Image Source: Wikipedia (https://en.wikipedia.org/wiki/Modbus)

A Modbus command includes the Modbus address of the intended target device. All devices receive the command, but only the device with the specified address executes the command and responds (with the exception of address 0, which is reserved for broadcast commands. Devices receiving a broadcast command will execute it but will not send a response).
All Modbus commands include a checksum to ensure the command has not been corrupted during transmission. Basic Modbus commands can instruct an RTU to modify a value in one of its registers, control or read an I/O port, or command a device to return data from one or more of its registers.

Many modems and gateways support the Modbus protocol because it is simple and easy to replicate. Some are specifically designed for this protocol. There are various implementations using wired, wireless communication, and even SMS or GPRS. However, designers must address challenges such as high latency and timing issues.

1.4 Functions Table

This section will briefly introduce the Modbus API usage list.

1.4.1 Modbus Master

The Modbus Master enables the 86Duino to emulate a Modbus Master and send packets to the Slave node on the channel.

Function Name	Description
`begin()`	Initialize the ModbusMaster object and specify it as Serial class or IPAddress class.
`attach()`	Initialize the ModbusMasterNode object and specify the mounting channel and corresponding node number.
`getResponseBuffer()`	Reads the value of the private array readData in the ModbusMasterNode class. readData is the array stored after reading back to the Slave device registers using the Read function code.
`clearResponseBuffer()`	Clears the value of the private array readData in the ModbusMasterNode class. readData is the array stored after reading back to the Slave device registers using the Read function code.
`setTransmitBuffer()`	Sets the value of the private array writeData in the ModbusMasterNode class. writeData is the content of the Modbus packet using the Write function code.
`clearTransmitBuffer()`	Clears the value of the private array writeData in the ModbusMasterNode class. writeData is the content of the Modbus packet using the Write function code.
`beginTransmission()`	Set the starting address of the register of the Slave device you want to write to, and use it with unquoted writeMultipleCoils or writeMultipleRegisters.
`sendBit()`	Set the quotes in bits to the private array writeData in the ModbusMasterNode class and stack the data automatically with beginTransmission and writeMultipleCoils without parameters. writeData is the contents of a Modbus packet using the Write function code.
`send()`	Set the quotes to the private array writeData in the ModbusMasterNode class and stack the data automatically with beginTransmission and writeMultipleRegisters without parameters. writeData is the contents of a Modbus packet using the Write function code.
`available()`	Return the data number of the private array readData in the ModbusMasterNode class.

	readData is the array stored after reading back to the Slave device registers using the Read function code.
`receive()`	Returns the data of the private array readData in the ModbusMasterNode class. readData is the array stored after reading back to the Slave device registers using the Read function code.
`readCoils()`	Send a packet read command to the Slave node to read the Coils register.
`readDiscreteInputs()`	Sends a packet command to the Slave node to read the Discrete Inputs register.
`readHoldingRegisters()`	Send a packet command to the Slave node to read the Holding registers.
`readInputRegisters()`	Send a packet command to the Slave node to read the Input registers.
`writeSingleCoil()`	Send a packet command to the Slave node to write to a single Coil register.
`writeSingleRegister()`	Send a packet command to the Slave node to write to a single Holding register.
`writeMultipleCoils()`	Send a packet command to the Slave node to write to multiple Coils registers.
`writeMultipleRegisters()`	Send a packet command to the Slave node to write to multiple Holding registers.
`maskWriteRegister()`	Send the packet mask operation instruction written to the Holding register to the Slave node with the following formula: (Current_Contents AND and_mask) OR (or_mask AND (NOT and_mask))
`readWriteMultipleRegisters()`	Send a packet command to the Slave node to read and write the Holding registers.

1.4.2 Modbus Slave

Modbus Slave enables the 86Duino to emulate a Modbus Slave node, receive commands and execute callback functions on the channel.

Function Name	Description
`begin()`	Initialize the ModbusSlave object and specify the mode and channel.
`attach()`	Initialize the ModbusSlaveNode object and specify the mount channel and node number.
`poll()`	Receives and parses Modbus packets and accesses the register and callback function according to the command.
`*cbFunc[]()`	cbFunc is a callback function pointer array. When poll() receives the corresponding function code, it will call the corresponding function pointed to in the cbFunc array.
`readCoil()`	Reads the Coils register.
`writeCoil()`	Write to the Coils register.
`writeDiscreteInput()`	Write to the Discrete Inputs register.
`readHoldingRegister()`	Reads the Holding register.
`writeHoldingRegister()`	Write to the Holding register.
`writeInputRegister()`	Write to the Input register.

1.4.3 Modbus Gateway

The Modbus Gateway enables the 86Duino to emulate the Modbus Gateway, a gateway for forwarding Modbus Master communication data.

Function Name	Description
`begin()`	Initialize the ModbusGateway object and specify the communication mode between this object and the Modbus Master.
`connect()`	Create a link between this gateway and the ModbusMasterNode object, which will correspond to a channel and node number.
`setTimeout()`	Set the timeout.
`poll()`	Receives and parses Modbus packets and sends them to the corresponding node according to the packet address.

Ch. 2

Functions

86Duino Coding IDE 500+.

Modbus Library

2.1 Modbus Master

The Modbus Master enables the 86Duino to emulate a Modbus Master and send packets to the Slave node on the channel.

Groups:

- ModbusMaster Class
- ModbusMasterNode Class

2.1.1 ModbusMaster Class

Use Serial / Ethernet for Modbus transfer.

Functions:

- begin()

The following APIs are still under development and are not recommended for use:

```
/*************************************************************************
    *  The following APIs are still under development and are not          *
    *  recommended for use.                                                *
    *************************************************************************/
uint8_t readCoils(uint8_t slave_id, uint16_t address, uint16_t size, uint16_t *data);
uint8_t readDiscreteInputs(uint8_t slave_id, uint16_t address, uint16_t size, uint16_t *data);
uint8_t readHoldingRegisters(uint8_t slave_id, uint16_t address, uint16_t size, uint16_t *data);
uint8_t readInputRegisters(uint8_t slave_id, uint16_t address, uint16_t size, uint16_t *data);
uint8_t writeSingleCoil(uint8_t slave_id, uint16_t address, uint16_t value);
uint8_t writeSingleRegister(uint8_t slave_id, uint16_t address, uint16_t value);
uint8_t writeMultipleCoils(uint8_t slave_id, uint16_t address, uint16_t size, uint16_t *data);
uint8_t writeMultipleRegisters(uint8_t slave_id, uint16_t address, uint16_t size, uint16_t *data);
uint8_t maskWriteRegister(uint8_t slave_id, uint16_t address, uint16_t and_mask, uint16_t or_mask);
uint8_t readWriteMultipleRegisters(uint8_t slave_id, uint16_t write_address, uint16_t write_size,
uint16_t *write_data, uint16_t read_address, uint16_t read_size, uint16_t *read_data);
```

begin()

Description

Initialize the ModbusMaster object and specify it as Serial class or IPAddress class.

Syntax

`bus1.begin(mode, serial)`

`bus2.begin(mode, modbus_server_ip)`

Parameters

- bus1/bus2: ModbusMaster object.
- mode: Specify the communication mode of the ModbusMaster object, which is coded MODBUS_RTU, MODBUS_ASCII, or MODBUS_TCP.
- Serial: Specify the Serial class as the transmission channel.
- modbus_server_ip: Specify the IPAddress class as the transport channel.

Returns

bool: return true if successful, return false if not.

Example

```
#include <Ethernet.h>
#include <Modbus.h>

ModbusMaster bus1;
ModbusMaster bus2;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 101);
IPAddress serverIp(192, 168, 1, 102);

void setup() {
  Serial485.begin(115200);
  bus1.begin(MODBUS_RTU, Serial485);
  // bus1.begin(MODBUS_ASCII, Serial485);
  Ethernet.begin(mac, localIp);
  bus2.begin(MODBUS_TCP, serverIp);
}

void loop() {
  //...
}
```

2.1.2 ModbusMasterNode Class

Send the corresponding node packet command.

Functions:

- attach()
- getResponseBuffer()
- clearResponseBuffer()
- setTransmitBuffer()
- clearTransmitBuffer()
- beginTransmission()
- sendBit()
- send()
- available()
- receive()
- readCoils()
- readDiscreteInputs()
- readHoldingRegisters()
- readInputRegisters()
- writeSingleCoil()
- writeSingleRegister()
- writeMultipleCoils()
- writeMultipleRegisters()
- maskWriteRegister()
- readWriteMultipleRegisters()

attach()

Description

Initialize the ModbusMasterNode object and specify the mounting channel and corresponding node number.

Syntax

```
node1.attach(slave_id, bus1)
node2.attach(bus2)
```

Parameters

- node1/node2: ModbusMasterNode object.
- slave_id: Specify the corresponding Slave node number; not specifying the number in TCP mode will make the number default to MODBUS_TCP_SLAVE (0xFF).
- bus1/bus2: Specify the ModbusMaster object to be mounted.

Returns

bool: return true if successful, return false if not.

Example

```
#include <Ethernet.h>
#include <Modbus.h>

ModbusMaster bus1;
ModbusMaster bus2;

ModbusMasterNode node1;
ModbusMasterNode node2;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 101);
IPAddress serverIp(192, 168, 1, 102);

void setup() {
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  // bus1.begin(MODBUS_ASCII, Serial485);

  Ethernet.begin(mac, localIp);
  bus2.begin(MODBUS_TCP, serverIp);

  node1.attach(16, bus1);
  node2.attach(bus2);
  // node2.attach(MODBUS_TCP_SLAVE, bus2);
```

```
}

void loop() {
  //...
}
```

getResponseBuffer()

Description

Reads the value of the private array readData in the ModbusMasterNode class.
readData is the array stored after reading back to the Slave device registers using the Read function code.

Syntax

`node1.getResponseBuffer(index, mode)`

Parameters

- node1: ModbusMasterNode object.
- index: index value of the readData array.
- mode: mode of reading data from readData array, its citation is `MODBUS_DATAMODE_BIT` or `MODBUS_DATAMODE_UINT16`. If no citation is given it will be `MODBUS_DATAMODE_UINT16` by default.

Returns

int: The value taken from the readData array, or 0xFFFF if it is wrong.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;
uint32_t receiveData;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readCoils(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("readCoils => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Coil Status receiveData: ");
    receiveData = node1.getResponseBuffer(0) | (node1.getResponseBuffer(1) <<
16);
```

```
    Serial.println(receiveData);
  }
}

void loop() {
  //...
}
```

clearResponseBuffer()

Description

Clears the value of the private array readData in the ModbusMasterNode class.
readData is the array stored after reading back to the Slave device registers using the Read function code.

Syntax

`node1.clearResponseBuffer()`

Parameters

- node1: ModbusMasterNode object.

Returns

None.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;
uint32_t receiveData;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readCoils(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("readCoils => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Coil Status receiveData: ");
    receiveData = node1.getResponseBuffer(0) | (node1.getResponseBuffer(1) <<
16);
    Serial.println(receiveData);
  }

  delay(1000);
```

```
  node1.clearResponseBuffer();
}

void loop() {
  //...
}
```

setTransmitBuffer()

Description

Sets the value of the private array writeData in the ModbusMasterNode class.
writeData is the content of the Modbus packet using the Write function code.

Syntax

`node1.setTransmitBuffer(index, value, mode)`

Parameters

- node1: ModbusMasterNode object.
- index: the index value of the writeData array.
- value: the value to be filled in the writeData array, i.e. the content of the Modbus packet to be sent.
- mode: mode of reading data from writeData array, the citation is `MODBUS_DATAMODE_BIT` or `MODBUS_DATAMODE_UINT16`, if no citation is given it will be `MODBUS_DATAMODE_UINT16` by default.

Returns

Int: If successful, return `MODBUS_SUCCESS`; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  node1.setTransmitBuffer(0, true, MODBUS_DATAMODE_BIT);
  node1.setTransmitBuffer(1, false, MODBUS_DATAMODE_BIT);
  result = node1.writeMultipleCoils(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: ");
    Serial.println(result);
  }
}
```

```
void loop() {
  //...
}
```

clearTransmitBuffer()

Description

Clears the value of the private array writeData in the ModbusMasterNode class.
writeData is the content of the Modbus packet using the Write function code.

Syntax

`node1.clearTransmitBuffer()`

Parameter

- node1: ModbusMasterNode object.

Returns

None.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;
uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);
  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  node1.setTransmitBuffer(0, true, MODBUS_DATAMODE_BIT);
  node1.setTransmitBuffer(1, false, MODBUS_DATAMODE_BIT);
  result = node1.writeMultipleCoils(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: ");
    Serial.println(result);
  }

  delay(1000);
  node1.clearTransmitBuffer();
}

void loop() {
  //...
}
```

beginTransmission()

Description

Set the starting address of the register of the Slave device you want to write to, and use it with unquoted writeMultipleCoils or writeMultipleRegisters.

Syntax

`node1.beginTransmission(write_address)`

Parameters

- node1: ModbusMasterNode object.
- write_address: The starting address of the register of the Slave device you want to write to.

Returns

None.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t reg[2];
uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  reg[0] = 0x1234;
  reg[1] = 0xabcd;
  node1.beginTransmission(5);
  node1.send(reg[0]);
  node1.send(reg[1]);
  result = node1.writeMultipleRegisters();

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: ");
    Serial.println(result);
  }
}
```

```
void loop() {
  //...
}
```

sendBit()

Description

Set the quotes in bits to the private array writeData in the ModbusMasterNode class and stack the data automatically with beginTransmission and writeMultipleCoils without parameters.
writeData is the contents of a Modbus packet using the Write function code.

Syntax

`node1.sendBit(value)`

Parameters

- node1: ModbusMasterNode object.
- value: Fill in the boolean value of the writeData array, which is the content of the Modbus packet to be sent.

Returns

None.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;
uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);
  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  node1.beginTransmission(5);
  node1.sendBit(true);
  node1.sendBit(false);
  result = node1.writeMultipleCoils();

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: "); Serial.println(result);
  }
}

void loop() {
  //...
}
```

send()

Description

Set the quotes to the private array writeData in the ModbusMasterNode class and stack the data automatically with beginTransmission and writeMultipleRegisters without parameters.
writeData is the contents of a Modbus packet using the Write function code.

Syntax

`node1.send(value)`

Parameters

- node1: ModbusMasterNode object.
- value: Fill in the writeData array with the value of the Modbus packet content to be sent, either `uint8_t`, `uint16_t` or `uint32_t`.

Returns

None.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t reg[2];
uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  reg[0] = 0x1234;
  reg[1] = 0xabcd;
  node1.beginTransmission(5);
  node1.send(reg[0]);
  node1.send(reg[1]);
  result = node1.writeMultipleRegisters();

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: "); Serial.println(result);
  }
}
```

```
void loop() {
  //...
}
```

available()

Description

Return the number of bytes of the private array readData in the ModbusMasterNode class. readData is the array stored after reading back to the Slave device registers using the Read function code.

Syntax

`node1.available()`

Parameters

- node1: ModbusMasterNode object.

Returns

uint8_t: Return the number of bytes of the private array readData in the ModbusMasterNode class.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;

uint8_t j = 0;
uint16_t data[6];

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readHoldingRegisters(0, 6);
  if (result != MODBUS_SUCCESS) {
    Serial.print("readHoldingRegisters => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Holding Register receiveData: ");
    while (node1.available() > 0) {
      data[j++] = node1.receive();
      Serial.println(data[j - 1]);
    }
  }
```

```
}

void loop() {
  //...
}
```

receive()

Description

Receive the data of the private array readData in the ModbusMasterNode class.
readData is the array stored after reading back to the Slave device registers using the Read function code.

Syntax

`node1.receive()`

Parameters

- node1: ModbusMasterNode object.

Returns

uint16_t: The data of the private array readData in the ModbusMasterNode class.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;

uint8_t j = 0;
uint16_t data[6];

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readHoldingRegisters(0, 6);
  if (result != MODBUS_SUCCESS) {
    Serial.print("readHoldingRegisters => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Holding Register receiveData: ");
    while (node1.available() > 0) {
      data[j++] = node1.receive();
      Serial.println(data[j - 1]);
    }
  }
```

```
}

void loop() {
  //...
}
```

readCoils()

Description

Send a packet read command to the Slave node to read the Coils register.

Syntax

`node1.readCoils(read_address, read_size)`

Parameters

- node1: ModbusMasterNode object.
- read_address: The starting address of the Coils register to be read.
- read_size: Size of the Coils register to be read.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readCoils(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("readCoils => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Coil Status receiveData: ");
    Serial.print(node1.getResponseBuffer(0, MODBUS_DATAMODE_BIT));
    Serial.print(", ");
    Serial.println(node1.getResponseBuffer(1, MODBUS_DATAMODE_BIT));
  }
}

void loop() {
```

```
  //...
}
```

readDiscreteInputs()

Description

Sends a packet command to the Slave node to read the Discrete Inputs register.

Syntax

`node1.readDiscreteInputs(read_address, read_size)`

Parameters

- node1: ModbusMasterNode Object.
- read_address: The starting address of the Discrete Inputs register to be read.
- read_size: The size of the Discrete Inputs registers to be read.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readDiscreteInputs(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("readDiscreteInputs => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Input Status receiveData: ");
    Serial.print(node1.getResponseBuffer(0, MODBUS_DATAMODE_BIT));
    Serial.print(", ");
    Serial.println(node1.getResponseBuffer(1, MODBUS_DATAMODE_BIT));
  }
}

void loop() {
```

```
  //...
}
```

readHoldingRegisters()

Description

Send a packet command to the Slave node to read the Holding registers.

Syntax

`node1.readHoldingRegisters(read_address, read_size)`

Parameters

- node1: ModbusMasterNode Object.
- read_address: The starting address of the Holding registers to be read.
- read_size: The size of the Holding registers to be read.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;
uint32_t receiveData;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readHoldingRegisters(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("readHoldingRegisters => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Holding Register receiveData: ");
    receiveData = node1.getResponseBuffer(0)
                | (node1.getResponseBuffer(1) << 16);
    Serial.println(receiveData);
  }
}
```

```
void loop() {
  //...
}
```

readInputRegisters()

Description

Send a packet command to the Slave node to read the Input registers.

Syntax

`node1.readInputRegisters(read_address, read_size)`

Parameters

- node1: ModbusMasterNode Object.
- read_address: The starting address of the Input registers to be read.
- read_size: The size of the Input registers to be read.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;
uint32_t receiveData;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.readInputRegisters(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("readInputRegisters => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Input Register receiveData: ");
    receiveData = node1.getResponseBuffer(0)
                | (node1.getResponseBuffer(1) << 16);
    Serial.println(receiveData);
  }
}
```

```
void loop() {
  //...
}
```

writeSingleCoil()

Description

Send a packet command to the Slave node to write to a single Coil register.

Syntax

`node1.writeSingleCoil(write_address, value)`

Parameters

- node1: ModbusMasterNode Object.
- write_address: The address of the Coil register you want to write to.
- value: The value of the Coil registers to write to, usually 0 or 1.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;
uint8_t led = 1;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.writeSingleCoil(5, led);

  if (result != MODBUS_SUCCESS) {
    Serial.print("writeSingleCoil => ErrorCode: ");
    Serial.println(result);
  }
}

void loop() {
  //...
}
```

writeSingleRegister()

Description

Send a packet command to the Slave node to write to a single Holding register.

Syntax

`node1.writeSingleRegister(write_address, write_size)`

Parameters

- node1: ModbusMasterNode Object.
- write_address: The address of the Holding Register to be written to.
- write_size: The size of the Holding Register to be written to.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;
uint16_t value = 0x12AB;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.writeSingleRegister(5, value);

  if (result != MODBUS_SUCCESS) {
    Serial.print("writeSingleRegister => ErrorCode: ");
    Serial.println(result);
  }
}

void loop() {
  //...
}
```

writeMultipleCoils()

Description

Send a packet command to the Slave node to write to multiple Coils registers.

Syntax

`node1.writeMultipleCoils(write_address, write_size)`

`node2.writeMultipleCoils()`

(See the example to illustrate the different functions and usage methods with or without parameters)

Parameters

- node1/node2: ModbusMasterNode Object.
- write_address: The starting address of the Coils register you want to write to.
- write_size: The size of the Coils register that you want to write to.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

Example 1: with parameters.

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;
uint8_t led = 1;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.writeSingleCoil(5, led);

  if (result != MODBUS_SUCCESS) {
    Serial.print("writeSingleCoil => ErrorCode: ");
    Serial.println(result);
  }
}

void loop() {
```

```
  //...
}
```

Example 2: without parameters.

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  node1.beginTransmission(5);
  node1.sendBit(true);
  node1.sendBit(false);
  result = node1.writeMultipleCoils();

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: ");
    Serial.println(result);
  }
}

void loop() {
  //...
}
```

writeMultipleRegisters()

Description

Send a packet command to the Slave node to write to multiple Holding Registers.

Syntax

`node1.writeMultipleRegister(write_address, write_size)`

`node2.writeMultipleRegister()`

(See the example to illustrate the difference in function and usage with or without parameters.)

Parameters

- node1/node2: ModbusMasterNode Object.
- write_address: The starting address of the Holding Registers you want to write to.
- write_size: The size of the Holding Registers to be written to.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

Example 1: with parameters.

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t reg[2];
uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  reg[0] = 0x1234;
  reg[1] = 0xabcd;
  node1.setTransmitBuffer(0, reg[0], MODBUS_DATAMODE_UINT16);
  node1.setTransmitBuffer(1, reg[1], MODBUS_DATAMODE_UINT16);
  result = node1.writeMultipleRegisters(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: ");
    Serial.println(result);
  }
```

```
}

void loop() {
  //...
}
```

Example 2: without parameters.

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t reg[2];
uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  reg[0] = 0x1234;
  reg[1] = 0xabcd;
  node1.beginTransmission(5);
  node1.send(reg[0]);
  node1.send(reg[1]);
  result = node1.writeMultipleRegisters();

  if (result != MODBUS_SUCCESS) {
    Serial.print("ErrorCode: ");
    Serial.println(result);
  }
}

void loop() {
  //...
}
```

maskWriteRegister()

Description

Send the packet mask operation instruction written to the Holding register to the Slave node with the following formula:

(Current_Contents AND and_mask) OR (or_mask AND (NOT and_mask))

Syntax

`node1.maskWriteRegister(write_address , and_mask , or_mask)`

Parameters

- node1: ModbusMasterNode Object.
- write_address: The address of the Holding Register to be written to.
- and_mask: The “with” gate mask parameter.
- or_mask: “or” gate mask parameter.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

#define AND_MASK    0xF87F
#define OR_MASK     0x0300

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t result;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  result = node1.maskWriteRegister(5, AND_MASK, OR_MASK);

  if (result != MODBUS_SUCCESS) {
    Serial.print("maskWriteRegister => ErrorCode: ");
    Serial.println(result);
  }
}
```

```
void loop() {
  //...
}
```

readWriteMultipleRegisters()

Description

Send a packet command to the Slave node to read and write the Holding Registers.

Syntax

`node1.readWriteMultipleRegisters(read_address, read_size, write_address, write_size)`

`node2.readWriteMultipleRegisters(read_address, read_size)`

(See the examples to illustrate the different functions and usage of the parameters)

Parameters

- node1/node2: ModbusMasterNode Object.
- read_address: The starting address of the Holding Registers to be read.
- read_size: The size of the Holding Registers to be read.
- write_address: The starting address of the Holding Registers you want to write to.
- write_size: The size of the Holding Registers that you want to write to.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

Example 1: with parameters.

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t reg[2];
uint8_t result;
uint32_t receiveData;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  reg[0] = 0x1234;
  reg[1] = 0xabcd;
  node1.setTransmitBuffer(0, reg[0], MODBUS_DATAMODE_UINT16);
  node1.setTransmitBuffer(1, reg[1], MODBUS_DATAMODE_UINT16);
  result = node1.readWriteMultipleRegisters(5, 2, 10, 2);
```

```
  if (result != MODBUS_SUCCESS) {
    Serial.print("readWriteMultipleRegisters => ErrorCode: ");
    Serial.println(result);
  } else {
    Serial.print("From Holding Register receiveData: ");
    receiveData = node1.getResponseBuffer(0)
                | (node1.getResponseBuffer(1) << 16);
    Serial.println(receiveData);
  }
}

void loop() {
  //...
}
```

Example 2: without parameters.

```
#include <Modbus.h>

ModbusMaster bus1;
ModbusMasterNode node1;

uint8_t reg[2];
uint8_t result;
uint32_t receiveData;

void setup() {
  Serial.begin(115200);
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  node1.attach(16, bus1);

  reg[0] = 0x1234;
  reg[1] = 0xabcd;
  node1.beginTransmission(10);
  node1.send(reg[0]);
  node1.send(reg[1]);
  result = node1.readWriteMultipleRegisters(5, 2);

  if (result != MODBUS_SUCCESS) {
    Serial.print("readWriteMultipleRegisters => ErrorCode: ");
    Serial.println(result);
```

```
  } else {
    Serial.print("From Holding Register receiveData: ");
    receiveData = node1.getResponseBuffer(0)
                | (node1.getResponseBuffer(1) << 16);
    Serial.println(receiveData);
  }
}

void loop() {
  //...
}
```

2.2 Modbus Slave

Modbus86 Slave enables the 86Duino to emulate a Modbus Slave node, receive commands and execute callback functions on the channel.

Groups:

- ModbusSlave Class
- ModbusSlaveNode Class

2.2.1 ModbusSlave Class

Use Serial / Ethernet for Modbus transfer.

Functions:

- begin()

begin()

Description

Initialize the ModbusSlave object and specify the mode and channel.

Syntax

```
bus1.begin(mode, serial)
bus2.begin(mode)
```

Parameters

- bus1/bus2: ModbusSlave object.
- mode: Specify the communication mode of the ModbusSlave object with MODBUS_RTU, MODBUS_ASCII, or MODBUS_TCP pins.
- serial: When mode is specified as MODBUS_RTU or MODBUS_ASCII, Serial class must be specified as the transmission channel.

Returns

bool: return true if successful, return false if not.

Example

```
#include <Ethernet.h>
#include <Modbus.h>

ModbusSlave bus1;
ModbusSlave bus2;

void setup() {
  Serial485.begin(115200);

  bus1.begin(MODBUS_RTU, Serial485);
  // bus1.begin(MODBUS_ASCII, Serial485);

  Ethernet.begin();
  bus2.begin(MODBUS_TCP);

}

void loop() {
  //...
}
```

2.2.2 ModbusSlaveNode Class

Receives and parses the corresponding packets and accesses the data registers and callback functions as instructed.

Functions:

- attach()
- poll()

The callback function with array of indicators.

- *cbFunc[]()

Direct access to the Slave register's function, is usually used in the callback function.

- readCoil()
- writeCoil()
- writeDiscreteInput()
- readHoldingRegister()
- writeHoldingRegister()
- writeInputRegister()

attach()

Description

Initialize the ModbusSlaveNode object and specify the mount channel and node number.

Syntax

```
node1.attach(slave_id, bus1)
node2.attach(bus2)
```

Parameters

- node1/node2: ModbusSlaveNode Object.
- slave_id: Specify the Slave node number, not specifying the number in TCP mode will cause the number to default to MODBUS_TCP_SLAVE (0xFF).
- bus1/bus2: Specify the ModbusSlave object to be mounted.

Returns

bool: return true if successful, return false if not.

Example

```
#include <Ethernet.h>
#include <Modbus.h>

ModbusSlave bus1;
ModbusSlave bus2;

ModbusSlaveNode node1;
ModbusSlaveNode node2;

void setup() {
  Serial485.begin(115200);
  bus1.begin(MODBUS_RTU, Serial485);
  // bus1.begin(MODBUS_ASCII, Serial485);

  Ethernet.begin();
  bus2.begin(MODBUS_TCP);

  node1.attach(16, bus1);
  node2.attach(bus2);
  // node2.attach(MODBUS_TCP_SLAVE, bus2);
}

void loop() {
  //...
}
```

poll()

Description

Receives and parses Modbus packets and accesses the register and callback function according to the command.

When the function code is Read command, it will call the callback function first and then read the register back; when the function code is Write command, it will write the register first and then call the callback function; when the function code is ReadWrite command, it will write the register first and then call the Write callback function, then call the Read callback function and finally read the register back.

Syntax

```
node1.poll()
node2.poll(buf, size)
```

Parameters

- node1/node2: ModbusSlaveNode object.
- buf: User-defined registry space.
- size: The size of the self-defined space.

Returns

Int: The length of the received Modbus packets.

Example

```
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

uint8_t write_single_coil( uint8_t function, uint16_t address, uint16_t
length) {
  uint16_t value;

  if (address == 0) {
    node.readCoil(address, 1, &value);
    if (value)
      digitalWrite(LED_BUILTIN, HIGH);
    else
      digitalWrite(LED_BUILTIN, LOW);
  }

  return MODBUS_SUCCESS;
}

void setup() {
```

```
  pinMode(LED_BUILTIN, OUTPUT);

  Serial485.begin(115200);

  /* Modbus RTU Mode via RS485. */
  bus.begin(MODBUS_RTU, Serial485);

  /* Slave node with ID 11. */
  node.attach(11, bus);

  /* Set the callback function of Write Single Coil (0x05). */
  node.cbFunc[MODBUS_CB_WRITE_SINGLE_COIL] = write_single_coil;
}

void loop() {
  node.poll();
}
```

*cbFunc[]()

Description

cbFunc is a callback function pointer array. When poll() receives the corresponding function code, it will call the corresponding function pointed to in the cbFunc array.

Syntax

```
uint8_t Func1(function, address, length) { return EXCEPTION_CODE; }
node1.cbFunc[MODBUS_CB_CODES] = Func1;
```

Parameters

- node1: ModbusSlaveNode object.
- Func1: User-defined function that returns a Byte of EXCEPTION_CODE.
- function: Modbus packet function code.
- address: The address of the bit that the Modbus packet wants to access.
- length: The length of the space that the Modbus packet wants to access.
- cbFunc[]: Call-back function letter array.
- MODBUS_CB_CODES: The callback function has a column index value and supports the following function codes.
  - MODBUS_CB_READ_COILS
  - MODBUS_CB_READ_DISCRETE_INPUTS
  - MODBUS_CB_READ_HOLDING_REGISTERS
  - MODBUS_CB_READ_INPUT_REGISTERS
  - MODBUS_CB_WRITE_SINGLE_COIL
  - MODBUS_CB_WRITE_SINGLE_REGISTER
  - MODBUS_CB_WRITE_MULTIPLE_COILS
  - MODBUS_CB_WRITE_MULTIPLE_REGISTERS
  - MODBUS_CB_MASK_WRITE_REGISTER

Return

None.

Example

```
#include <Arduino.h>
#include <Ethernet.h>
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 102);

uint8_t modbus_callback(uint8_t function, uint16_t address, uint16_t length)
{
```

```
  Serial.print("In callback function with CMD: ");
  Serial.println(function);
  return MODBUS_SUCCESS;
}

void setup()
{
  Serial.begin(115200);
  while (!Serial);

  pinMode(LED_BUILTIN, OUTPUT);

  Ethernet.begin(mac, localIp);

  /* Modbus TCP Mode via Ethernet. */
  bus.begin(MODBUS_TCP);

  /* Slave node initialize. */
  node.attach(1, bus);

  node.cbFunc[MODBUS_CB_READ_COILS             ] = modbus_callback;
  node.cbFunc[MODBUS_CB_READ_DISCRETE_INPUTS    ] = modbus_callback;
  node.cbFunc[MODBUS_CB_READ_HOLDING_REGISTERS  ] = modbus_callback;
  node.cbFunc[MODBUS_CB_READ_INPUT_REGISTERS    ] = modbus_callback;
  node.cbFunc[MODBUS_CB_WRITE_SINGLE_COIL       ] = modbus_callback;
  node.cbFunc[MODBUS_CB_WRITE_SINGLE_REGISTER   ] = modbus_callback;
  node.cbFunc[MODBUS_CB_WRITE_MULTIPLE_COILS    ] = modbus_callback;
  node.cbFunc[MODBUS_CB_WRITE_MULTIPLE_REGISTERS] = modbus_callback;
  node.cbFunc[MODBUS_CB_MASK_WRITE_REGISTER     ] = modbus_callback;
}

void loop()
{
  node.poll();
}
```

readCoil()

Description

Reads the Coils register.

Syntax

`node1.readCoil(address)`

`node2.readCoil(address, size, buffer)`

Parameters

- node1/node2: ModbusSlaveNode object.
- address: The address of the Coils register you want to read.
- size: The number of Coils registers you want to read.
- buffer: Copy the read-out Coils register data to the array space pointed to by the buffer.

Returns

Int.

1. If only address is provided: Returns the value at the specified address, either `MODBUS_COIL_ON` or `MODBUS_COIL_OFF`.
2. If address, size, and buffer are provided: If successful, return `MODBUS_SUCCESS`; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

uint8_t write_single_coil( uint8_t function, uint16_t address, uint16_t
length) {
  uint16_t value;

  if (address == 0) {
    node.readCoil(address, 1, &value);
    if (value)
      digitalWrite(LED_BUILTIN, HIGH);
    else
      digitalWrite(LED_BUILTIN, LOW);
  }

  return MODBUS_SUCCESS;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
```

```
  Serial485.begin(115200);

  /* Modbus RTU Mode via RS485. */
  bus.begin(MODBUS_RTU, Serial485);

  /* Slave node with ID 11. */
  node.attach(11, bus);

  /* Set the callback function of Write Single Coil (0x05). */
  node.cbFunc[MODBUS_CB_WRITE_SINGLE_COIL] = write_single_coil;
}

void loop() {
  node.poll();
}
```

writeCoil()

Description

Write to the Coils register.

Syntax

```
node1.writeCoil(address, value)
node2.writeCoil(address, size, buffer)
```

Parameters

- node1/node2: ModbusSlaveNode object.
- address: Address of the Coils register to write to.
- value: The value of the single Coils register that you want to write to.
- size: The number of plural Coils registers to write to.
- buffer: Copy the array data in the array space pointed to by the buffer to the Coil register.

Returns

Int: If successful, return `MODBUS_SUCCESS`; otherwise, return EXCEPTION_CODE.

writeDiscreteInput()

Description

Write to the Discrete Inputs register.

Syntax

```
node1.writeDiscreteInput(address, value)
node2.writeDiscreteInput(address, size, buffer)
```

Parameters

- node1/node2: ModbusSlaveNode object.
- address: The address of the Discrete Inputs registers to write to.
- value: The value of the single Discrete Inputs register that you want to write to.
- size: The number of plural Discrete Inputs registers to write to.
- buffer: Copy the data of the array in the array space pointed to by the buffer to the Discrete Inputs register.

Returns

Int: If successful, return `MODBUS_SUCCESS`; otherwise, return EXCEPTION_CODE.

readHoldingRegister()

Description

Reads the Holding register.

Syntax

```
node1.readHoldingRegister(address)
node2.readHoldingRegister(address, size, buffer)
```

Parameters

- node1/node2: ModbusSlaveNode object.
- address: Address of the Holding Registers to be read.
- size: The number of Holding Registers to be read.
- buffer: Copy the read-out Holding Registers data to the array space pointed to by the buffer.

Returns

Int.

1. If only address is provided: Returns the Holding Register location data
2. If address, size, and buffer are provided: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

uint32_t command = 0;
uint32_t lasttime = 0;

uint8_t read_holding_registers( uint8_t function, uint16_t address, uint16_t
length) {
  for (int i = address; i < address + length; i++) {
    if (i == 16)
      node.writeHoldingRegister(i, command & 0xFFFF);
    else if (i == 17)
      node.writeHoldingRegister(i, (command >> 16) & 0xFFFF);
  }

  return MODBUS_SUCCESS;
}

uint8_t write_multiple_registers( uint8_t function, uint16_t address,
uint16_t length) {
  int i;
```

```
  for (i = address; i < address + length; i++) {
    if (i == 16)
      command = (command & 0xFFFF0000) | node.readHoldingRegister(i);
    else if (i == 17)
      command =
        (command & 0x0000FFFF) | (node.readHoldingRegister(i) << 16);
  }

  return MODBUS_SUCCESS;
}

void setup() {
  Serial485.begin(115200);

  /* Modbus RTU Mode via RS485. */
  bus.begin(MODBUS_RTU, Serial485);

  /* Slave node with ID 11. */
  node.attach(11, bus);

  /* Set the callback function of Read Holding Registers (0x03). */
  node.cbFunc[MODBUS_CB_READ_HOLDING_REGISTERS] = read_holding_registers;

  /* Set the callback function of Write Multiple Registers (0x10). */
  node.cbFunc[MODBUS_CB_WRITE_MULTIPLE_REGISTERS] = write_multiple_registers;
}

void loop()
{
  uint32_t now;
  node.poll();

  now = millis();
  if (now - lasttime > 500) {
    lasttime = now;
    Serial.print("commnad = ");
    Serial.println(command);
  }
}
```

writeHoldingRegister()

Description

Write to the Holding Registers.

Syntax

```
node1.writeHoldingRegister(address, value)
node2.writeHoldingRegister(address, size, buffer)
```

Parameters

- node1/node2: ModbusSlaveNode object.
- address: Write to the Holding Registers.
- value: The value of the single Holding register that you want to write to.
- size: The number of plural Holding Register to write to.
- buffer: Copy the array data in the array space pointed to by the buffer to the Holding Registers.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

uint32_t command = 0;
uint32_t lasttime = 0;

uint8_t read_holding_registers( uint8_t function, uint16_t address, uint16_t
length) {
  for (int i = address; i < address + length; i++) {
    if (i == 16)
      node.writeHoldingRegister(i, command & 0xFFFF);
    else if (i == 17)
      node.writeHoldingRegister(i, (command >> 16) & 0xFFFF);
  }

  return MODBUS_SUCCESS;
}

uint8_t write_multiple_registers( uint8_t function, uint16_t address,
uint16_t length) {
  int i;
```

```
  for (i = address; i < address + length; i++) {
    if (i == 16)
      command = (command & 0xFFFF0000) | node.readHoldingRegister(i);
    else if (i == 17)
      command =
        (command & 0x0000FFFF) | (node.readHoldingRegister(i) << 16);
  }

  return MODBUS_SUCCESS;
}

void setup() {
  Serial485.begin(115200);

  /* Modbus RTU Mode via RS485. */
  bus.begin(MODBUS_RTU, Serial485);

  /* Slave node with ID 11. */
  node.attach(11, bus);

  /* Set the callback function of Read Holding Registers (0x03). */
  node.cbFunc[MODBUS_CB_READ_HOLDING_REGISTERS] = read_holding_registers;

  /* Set the callback function of Write Multiple Registers (0x10). */
  node.cbFunc[MODBUS_CB_WRITE_MULTIPLE_REGISTERS] = write_multiple_registers;
}

void loop()
{
  uint32_t now;

  node.poll();

  now = millis();
  if (now - lasttime > 500) {
    lasttime = now;
    Serial.print("commnad = ");
    Serial.println(command);
  }
}
```

writeInputRegister()

Description

Write to the Input Registers.

Syntax

```
node1.writeInputRegister(address, value)
node2.writeInputRegister(address, size, buffer)
```

Parameters

- node1/node2: ModbusSlaveNode object.
- address: The address of the Input Registers to write to.
- value: The value of the single Input Register that you want to write to.
- size: The number of plural Input Register you want to write to.
- buffer: Copy the data of the array in the array space pointed to by the buffer to the Input Registers.

Returns

Int: If successful, return MODBUS_SUCCESS; otherwise, return EXCEPTION_CODE.

Example

```
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

uint8_t read_input_registers(uint8_t function, uint16_t address, uint16_t
length) {
  int sensorPin = 2; // A2

  if (address <= 2 && address + length > 2)
    node.writeInputRegister(2, analogRead(sensorPin));

  return MODBUS_SUCCESS;
}

void setup() {
  Serial485.begin(115200);

  /* Modbus RTU Mode via RS485. */
  bus.begin(MODBUS_RTU, Serial485);

  /* Slave node with ID 11. */
  node.attach(11, bus);
```

```
  /* Set the callback function of Read Input Registers (0x04). */
  node.cbFunc[MODBUS_CB_READ_INPUT_REGISTERS] = read_input_registers;
}

void loop() {
  node.poll();
}
```

2.3 Modbus Gateway

The Modbus Gateway enables the 86Duino to emulate the Modbus Gateway, a gateway for forwarding Modbus Master communication data.

Groups:

- ModbusGateway Class

2.3.1 ModbusGateway Class

A gateway that forwards Modbus Master communication data.

Functions:

- begin()
- connect()
- setTimeout()
- poll()

begin()

Description

Initialize the ModbusGateway object and specify the communication mode between this object and the Modbus Master.

Syntax

```
gateway1.begin(mode, serial)
gateway2.begin(mode)
```

Parameters

- gateway1/gateway2: ModbusGateway object.
- mode: Specify the communication mode between the ModbusGateway object and the Modbus Master with `MODBUS_RTU`, `MODBUS_ASCII` or `MODBUS_TCP` quotes.
- serial: Specify the Serial class as the transmission channel.

Returns

bool: Returns true if successful, false if not

Example

```
#include <Modbus.h>
#include <Ethernet.h>

ModbusGateway gateway1;
ModbusGateway gateway2;

void setup()
{
  Serial485.begin(115200);
  gateway1.begin(MODBUS_RTU, Serial485);

  Ethernet.begin();
  gateway2.begin(MODBUS_TCP);
}

void loop() {
  // ...
}
```

connect()

Description

Create a link between this gateway and the ModbusMasterNode object, which will correspond to a channel and node number.

Syntax

`gateway.connect(node)`

Parameters

- gateway: ModbusGateway object.
- node: ModbusMasterNode object, the object will correspond to a channel and node number.

Returns

bool: Returns true if successful, false if not.

Example

```
#include <Modbus.h>
#include <Ethernet.h>

ModbusMaster bus;
ModbusMasterNode node1;
ModbusMasterNode node2;

ModbusGateway gateway;

void setup() {
  Ethernet.begin();
  Serial485.begin(115200);
  gateway.begin(MODBUS_TCP);
  bus.begin(MODBUS_RTU, Serial485);

  node1.attach(11, bus);
  node2.attach(12, bus);

  gateway.connect(node1);
  gateway.connect(node2);
}

void loop() {
  gateway.poll();
}
```

setTimeout()

Description

Set the timeout.

Syntax

`gateway.setTimeout(timeout)`

Parameters

- gateway: ModbusGateway object.
- timeout: Set the timeout time in milliseconds. During the poll phase, if the timeout expires, the EXCEPTION_CODE will be returned to the Modbus Master.

Return

None.

Example

```
#include <Modbus.h>
#include <Ethernet.h>

ModbusMaster bus;
ModbusMasterNode node1;
ModbusMasterNode node2;

ModbusGateway gateway;

void setup() {
  Ethernet.begin();
  Serial485.begin(115200);
  gateway.begin(MODBUS_TCP);
  bus.begin(MODBUS_RTU, Serial485);

  node1.attach(11, bus);
  node2.attach(12, bus);

  gateway.connect(node1);
  gateway.connect(node2);
  gateway.setTimeout(500);
}

void loop() {
  gateway.poll();
}
```

poll()

Description

Receives and parses Modbus packets and sends them to the corresponding node according to the packet address.

Syntax

`gateway.poll()`

Parameters

- gateway: ModbusGateway object.

Returns

int: The length of the received Modbus packets.

Example

```
#include <Modbus.h>
#include <Ethernet.h>

ModbusMaster bus;
ModbusMasterNode node1;
ModbusMasterNode node2;

ModbusGateway gateway;

void setup() {
  Ethernet.begin();
  Serial485.begin(115200);
  gateway.begin(MODBUS_TCP);
  bus.begin(MODBUS_RTU, Serial485);

  node1.attach(11, bus);
  node2.attach(12, bus);

  gateway.connect(node1);
  gateway.connect(node2);
  gateway.setTimeout(500);
}

void loop() {
  gateway.poll();
}
```

Ch. 3

Example

3.1 Modbus Master

The Modbus86 Master enables the 86Duino to emulate a Modbus Master and send packets to the Slave node on the channel.

3.1.1 Example: Modbus Master with RS485

Example of Modbus Master with RS485 enabled and using Modbus RTU as the communication method.

```
#include <Modbus.h>

ModbusMaster bus;
ModbusMasterNode node1;
ModbusMasterNode node2;

int led = 0;
uint32_t value = 0;

void setup() {
  Serial485.begin(115200);
  /* Modbus RTU Mode via RS485. */
  bus.begin(MODBUS_RTU, Serial485);
  /* Slave node 1 with ID 11. */
  node1.attach(11, bus);
  /* Slave node 2 with ID 12. */
  node2.attach(12, bus);
}

void loop() {
  uint16_t reg[2];
  /* Write 1 coil to address 0 of the slave with ID 11. */
  node1.writeSingleCoil(0, led);
  /* Write 2 word to holding registers address 16 of the slave with ID 11. */
  reg[0] = value & 0xFFFF;
  reg[1] = (value >> 16) & 0xFFFF;
  node1.setTransmitBuffer(0, reg[0]);
  node1.setTransmitBuffer(1, reg[1]);
  node1.writeMultipleRegisters(16, 2);
```

```
  /* Read 2 word from holding registers address 16 of the slave with ID 11.
*/
  node1.readHoldingRegisters(16, 2);
  Serial.print("From Node 1 Holding Register: ");
  value = node1.getResponseBuffer(0)
         | (node1.getResponseBuffer(1) << 16);
  Serial.println(value);
  /* Read 1 word from input registers address 2 of
     the slave with ID 11. */
  node1.readInputRegisters(2, 1);
  Serial.print("             Input Register: ");
  Serial.print(node1.getResponseBuffer(0));
  Serial.println();
  /* Write 1 coil to address 0 of the slave with ID 12. */
  node2.writeSingleCoil(0, led);
  /* Write 2 word to holding registers address 16 of the slave with ID 12. */
  reg[0] = value & 0xFFFF;
  reg[1] = (value >> 16) & 0xFFFF;
  node2.setTransmitBuffer(0, reg[0]);
  node2.setTransmitBuffer(1, reg[1]);
  node2.writeMultipleRegisters(16, 2);
  /* Read 2 word from holding registers address 16 of the slave with ID 12.
*/
  node2.readHoldingRegisters(16, 2);
  Serial.print("From Node 2 Holding Register: ");
  value = node2.getResponseBuffer(0)
         | (node2.getResponseBuffer(1) << 16);
  Serial.println(value);
  /* Read 1 word from input registers address 2 of the slave with ID 12. */
  node2.readInputRegisters(2, 1);
  Serial.print("             Input Register: ");
  Serial.print(node2.getResponseBuffer(0));
  Serial.println();

  led = !led;
  value++;

  delay(1000);
}
```

3.1.2 Example: Modbus Master using Ethernet

Example of a Modbus Master using Ethernet and Modbus TCP as the communication method.

```
#include <Ethernet.h>
#include <Modbus.h>

ModbusMaster bus;
ModbusMasterNode node;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 101);
IPAddress serverIp(192, 168, 1, 102);

int led = 0;
uint32_t value = 0;

void setup()
{
  Ethernet.begin(mac, localIp);

  /* Modbus TCP Mode via Ethernet. */
  bus.begin(MODBUS_TCP, serverIp);

  /* Slave node initialize. */
  node.attach(11, bus);
}

void loop()
{
  uint16_t reg[2];

  /* Write 1 coil to address 0 of the slave with ID 11. */
  node.writeSingleCoil(0, led);

  /* Write 2 word to holding registers address 16 of the slave with ID 11. */
  reg[0] = value & 0xFFFF;
  reg[1] = (value >> 16) & 0xFFFF;
  node.setTransmitBuffer(0, reg[0]);
  node.setTransmitBuffer(1, reg[1]);
  node.writeMultipleRegisters(16, 2);
```

```
  /* Read 2 word from holding registers address 16 of the slave with ID 11.
*/
  node.readHoldingRegisters(16, 2);
  Serial.print("From Node 1 Holding Register: ");
  value = node.getResponseBuffer(0)
         | (node.getResponseBuffer(1) << 16);
  Serial.println(value);

  /* Read 1 word from input registers address 2 of the slave with ID 11. */
  node.readInputRegisters(2, 1);
  Serial.print("             Input Register: ");
  Serial.print(node.getResponseBuffer(0));
  Serial.println();

  led = !led;
  value++;

  delay(1000);
}
```

3.2 Modbus Slave

Modbus86 Slave enables the 86Duino to emulate a Modbus Slave node, receive commands and execute callback functions on the channel.

3.2.1 Example: Modbus Slave with RS485

Example of a Slave device with RS485 enabled and using Modbus RTU as the communication method.

```
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

uint8_t write_single_coil( uint8_t function, uint16_t address, uint16_t
length) {
  uint16_t value;

  if (address == 0) {
    node.readCoil(address, 1, &value);
    if (value)
      digitalWrite(LED_BUILTIN, HIGH);
    else
      digitalWrite(LED_BUILTIN, LOW);
  }

  return MODBUS_SUCCESS;
}

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);

  Serial485.begin(115200);

  /* Modbus RTU Mode via RS485. */
  bus.begin(MODBUS_RTU, Serial485);

  /* Slave node with ID 11. */
  node.attach(11, bus);
```

```
  /* Set the callback function of Write Single Coil (0x05). */
  node.cbFunc[MODBUS_CB_WRITE_SINGLE_COIL] = write_single_coil;
}

void loop() {
  node.poll();
}
```

3.2.2 Example: Modbus Slave using Ethernet

Example of a Slave device with Ethernet-enabled and using Modbus TCP as the communication method.

```
#include <Arduino.h>
#include <Ethernet.h>
#include <Modbus.h>

ModbusSlave bus;
ModbusSlaveNode node;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 102);

uint8_t modbus_callback(uint8_t function, uint16_t address, uint16_t length)
{
  Serial.print("In callback function with CMD: ");
  Serial.println(function);
  return MODBUS_SUCCESS;
}

void setup()
{
  Serial.begin(115200);
  while (!Serial);

  pinMode(LED_BUILTIN, OUTPUT);

  Ethernet.begin(mac, localIp);

  /* Modbus TCP Mode via Ethernet. */
  bus.begin(MODBUS_TCP);

  /* Slave node initialize. */
  node.attach(1, bus);

  node.cbFunc[MODBUS_CB_READ_COILS             ] = modbus_callback;
  node.cbFunc[MODBUS_CB_READ_DISCRETE_INPUTS    ] = modbus_callback;
  node.cbFunc[MODBUS_CB_READ_HOLDING_REGISTERS  ] = modbus_callback;
  node.cbFunc[MODBUS_CB_READ_INPUT_REGISTERS    ] = modbus_callback;
  node.cbFunc[MODBUS_CB_WRITE_SINGLE_COIL       ] = modbus_callback;
```

```
  node.cbFunc[MODBUS_CB_WRITE_SINGLE_REGISTER   ] = modbus_callback;
  node.cbFunc[MODBUS_CB_WRITE_MULTIPLE_COILS    ] = modbus_callback;
  node.cbFunc[MODBUS_CB_WRITE_MULTIPLE_REGISTERS] = modbus_callback;
  node.cbFunc[MODBUS_CB_MASK_WRITE_REGISTER     ] = modbus_callback;
}

void loop()
{
  node.poll();
}
```

3.3 Modbus Gateway

The Modbus Gateway enables the 86Duino to emulate the Modbus Gateway, a gateway for forwarding Modbus Master communication data.

3.3.1 Example: Modbus TCP with Modbus Master

Use Modbus TCP as the communication method with Modbus Master and forward its communication data to the node on Modbus RTU.

```
#include <Modbus.h>
#include <Ethernet.h>

ModbusMaster bus;
ModbusMasterNode node1;
ModbusMasterNode node2;

ModbusGateway gateway;

byte mac[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
IPAddress localIp(192, 168, 1, 102);

void setup()
{
  Ethernet.begin(mac, localIp);
  Serial485.begin(115200);

  gateway.begin(MODBUS_TCP);

  bus.begin(MODBUS_RTU, Serial485);

  node1.attach(11, bus);
  node2.attach(12, bus);

  gateway.connect(node1);
  gateway.connect(node2);
  gateway.setTimeout(500);
}

void loop()
```

```
{
  gateway.poll();
}
```

Appendix

A.1 Exception Code

Description

The exception code is used for the Modbus Slave node to send back packets so that the Modbus Master can get feedback.

Code

- MODBUS_SUCCESS (0x00)
  - The packet was successfully transferred.
- MODBUS_ILLEGAL_FUNCTION (0x01)
  - The Slave device receives an unallowed function code.
- MODBUS_ILLEGAL_DATA_ADDRESS (0x02)
  - The Slave device receives an unallowed address space.
- MODBUS_ILLEGAL_DATA_VALUE (0x03)
  - The Slave device receives an unallowed value in the specified address space.
- MODBUS_SERVER_DEVICE_FAILURE (0x04)
  - A non-recoverable error occurred during operation.
- MODBUS_ACKNOWLEDGE (0x05)
  - The instruction has been accepted and is being processed, but it will take a long time to complete.
- MODBUS_SERVER_DEVICE_BUSY (0x06)
  - Long command being processed.
- MODBUS_MEMORY_PARITY_ERROR (0x08)
  - Parity error detected when trying to read memory.
- MODBUS_GATEWAY_PATH_UNAVAILABLE (0x0A)
  - Unavailable Modbus Gateway paths.
- MODBUS_GATEWAY_TARGET_DEVICE_FAILED_TO_RESPOND (0x0B)
  - Modbus Gateway devices are not on the network.
- MODBUS_INVALID_SLAVE_ID (0xE0)
  - Invalid Slave Number.
- MODBUS_INVALID_FUNCTION (0xE1)
  - Invalid Function Code.
- MODBUS_RESPONSE_TIMEOUT (0xE2)
  - Replies over time.
- MODBUS_INVALID_CRC (0xE3)
  - Ineffective cyclic redundancy checks.
- MODBUS_ILLEGAL_DATA_MODE (0xE4)
  - Illegal data patterns.
- MODBUS_INVALID_BUS (0xE5)
  - Ineffective access.

Warranty

This product is warranted to be in good working order for a period of one year from the date of purchase. Should this product fail to be in good working order at any time during this period, we will, at our option, replace or repair it at no additional charge except as set forth in the following terms. This warranty does not apply to products damaged by misuse, modifications, accident or disaster. Vendor assumes no liability for any damages, lost profits, lost savings or any other incidental or consequential damage resulting from the use, misuse of, originality to use this product. Vendor will not be liable for any claim made by any other related party. Return authorization must be obtained from the vendor before returned merchandise will be accepted. Authorization can be obtained by calling or faxing the vendor and requesting a Return Merchandise Authorization (RMA) number. Returned goods should always be accompanied by a clear problem description.