```cpp
#include "MgsModbusEth2.h"

// For Arduino 1.0
EthernetServer MbServer(MB_PORT);
EthernetClient MbmClient;


//#define DEBUG

MgsModbus::MgsModbus()
{
}

//****************** Send data for ModBusMaster ***************
bool MgsModbus::Req(MB_FC FC, word Ref, word Count, word Pos, byte IP[4])
{
  MbmFC = FC;
  byte ServerIp[4];
  ServerIp[0] = IP[0];
  ServerIp[1] = IP[1];
  ServerIp[2] = IP[2];
  ServerIp[3] = IP[3];
  MbmByteArray[0] = 0;  // ID high byte
  MbmByteArray[1] = 1;  // ID low byte
  MbmByteArray[2] = 0;  // protocol high byte
  MbmByteArray[3] = 0;  // protocol low byte
  MbmByteArray[5] = 6;  // Lenght low byte;
  MbmByteArray[4] = 0;  // Lenght high byte
  MbmByteArray[6] = 1;  // unit ID
  MbmByteArray[7] = FC; // function code
  MbmByteArray[8] = highByte(Ref);
  MbmByteArray[9] = lowByte(Ref);

  //****************** Read Coils (1) & Read Input discretes (2) **********************
  if(FC == MB_FC_READ_COILS || FC == MB_FC_READ_DISCRETE_INPUT) {
```

```
    if (Count < 1) {Count = 1;}
    if (Count > 125) {Count = 2000;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
  }
  //****************** Read Registers (3) & Read Input registers (4) ******************
  if(FC == MB_FC_READ_REGISTERS || FC == MB_FC_READ_INPUT_REGISTER) {
    if (Count < 1) {Count = 1;}
    if (Count > 125) {Count = 125;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
  }
  //****************** Write Coil (5) **********************
  if(MbmFC == MB_FC_WRITE_COIL) {
    if (GetBit(Pos)) {MbmByteArray[10] = 0xFF;} else {MbmByteArray[10] = 0;} // 0xFF coil on 0x00 coil off
    MbmByteArray[11] = 0; // always zero
  }
  //****************** Write Register (6) ******************
  if(MbmFC == MB_FC_WRITE_REGISTER) {
    MbmByteArray[10] = highByte(MbData[Pos]);
    MbmByteArray[11] = lowByte(MbData[Pos]);
  }
  //****************** Write Multiple Coils (15) **********************
  // not fuly tested
  if(MbmFC == MB_FC_WRITE_MULTIPLE_COILS) {
    if (Count < 1) {Count = 1;}
    if (Count > 800) {Count = 800;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
    MbmByteArray[12] = (Count + 7) /8;
    MbmByteArray[4] = highByte(MbmByteArray[12] + 7); // Lenght high byte
    MbmByteArray[5] = lowByte(MbmByteArray[12] + 7); // Lenght low byte;
    for (int i=0; i<Count; i++) {
      bitWrite(MbmByteArray[13+(i/8)],i-((i/8)*8),GetBit(Pos+i));
    }
```

```
  }
  //****************** Write Multiple Registers (16) ******************
  if(MbmFC == MB_FC_WRITE_MULTIPLE_REGISTERS) {
    if (Count < 1) {Count = 1;}
    if (Count > 100) {Count = 100;}
    MbmByteArray[10] = highByte(Count);
    MbmByteArray[11] = lowByte(Count);
    MbmByteArray[12] = (Count*2);
    MbmByteArray[4] = highByte(MbmByteArray[12] + 7); // Lenght high byte
    MbmByteArray[5] = lowByte(MbmByteArray[12] + 7); // Lenght low byte;
    for (int i=0; i<Count;i++) {
      MbmByteArray[(i*2)+13] = highByte (MbData[Pos + i]);
      MbmByteArray[(i*2)+14] = lowByte (MbData[Pos + i]);
    }
  }
  //****************** ?? ******************
  MbmClient.connect(ServerIp, 502);
  delay(10);
  if (MbmClient.connected()) {
    #ifdef DEBUG
      Serial.println("connected with modbus slave");
      Serial.print("Master request: ");
      for(int i=0;i<MbmByteArray[5]+6;i++) {
        if(MbmByteArray[i] < 16){Serial.print("0");}
        Serial.print(MbmByteArray[i],HEX);
        if (i != MbmByteArray[5]+5) {Serial.print(".");} else {Serial.println();}
      }
    #endif

    delay(5);

    for(int i=0;i<MbmByteArray[5]+6;i++) {
      MbmClient.write(MbmByteArray[i]);
    }
```

```
    MbmCounter = 0;
    MbmByteArray[7] = 0;
    MbmPos = Pos;
    MbmBitCount = Count;
    return true;

  } else {
    #ifdef DEBUG
      Serial.println("\nconnection with modbus slave failed ");
      Serial.print("Master request: ");
      for (int i = 0; i<MbmByteArray[5] + 6; i++) {
          if (MbmByteArray[i] < 16) { Serial.print("0"); }
          Serial.print(MbmByteArray[i], HEX);
          if (i != MbmByteArray[5] + 5) { Serial.print("."); }
          else { Serial.println(); }
      }
    #endif
    return false;
    MbmClient.stop();
    delay(10);
  }
}


//****************** Recieve data for ModBusMaster ****************
void MgsModbus::MbmRun()
{
  //****************** Read from socket ****************
    while (MbmClient.connected()&(MbmCounter<2000)) {           //changed from available()
    MbmByteArray[MbmCounter] = MbmClient.read();
    if (MbmCounter > 4)  {
      if (MbmCounter == MbmByteArray[5] + 5) { // the full answer is recieved
        MbmClient.stop();
        MbmProcess();
        #ifdef DEBUG
```

```
          Serial.println("Reading MODBUS Reply...");
        #endif
      }
    }
    MbmCounter++;
  }
}

void MgsModbus::MbmProcess()
{
  MbmFC = SetFC(int (MbmByteArray[7]));
  #ifdef DEBUG
  Serial.println("IN PROCESS BLOCK");
    for (int i=0;i<MbmByteArray[5]+6;i++) {
      if(MbmByteArray[i] < 16) {Serial.print("0");}
      Serial.print(MbmByteArray[i],HEX);
      if (i != MbmByteArray[5]+5) {Serial.print(".");
      } else {Serial.println();}
    }
  #endif
  //****************** Read Coils (1) & Read Input discretes (2) **********************
  if(MbmFC == MB_FC_READ_COILS || MbmFC == MB_FC_READ_DISCRETE_INPUT) {
    word Count = MbmByteArray[8] * 8;
    if (MbmBitCount < Count) {
      Count = MbmBitCount;
    }
    for (int i=0;i<Count;i++) {
      if (i + MbmPos < MbDataLen * 16) {
        SetBit(i + MbmPos,bitRead(MbmByteArray[(i/8)+9],i-((i/8)*8)));
      }
    }
  }
  //****************** Read Registers (3) & Read Input registers (4) *****************
  if(MbmFC == MB_FC_READ_REGISTERS || MbmFC == MB_FC_READ_INPUT_REGISTER) {
    #ifdef DEBUG
```

```
      Serial.println("Pushing data to storage register");
    #endif
    word Pos = MbmPos;
    for (int i=0;i<MbmByteArray[8];i=i+2) {
      if (Pos < MbDataLen) {
        MbData[Pos] = (MbmByteArray[i+9] * 0x100) + MbmByteArray[i+1+9];
        Pos++;
      }
    }
  }
  //****************** Write Coil (5) **********************
  if(MbmFC == MB_FC_WRITE_COIL){
  }
  //****************** Write Register (6) ******************
  if(MbmFC == MB_FC_WRITE_REGISTER){
  }
  //****************** Write Multiple Coils (15) ***********************
  if(MbmFC == MB_FC_WRITE_MULTIPLE_COILS){
  }
  //****************** Write Multiple Registers (16) ******************
  if(MbmFC == MB_FC_WRITE_MULTIPLE_REGISTERS){
  }
}


//****************** Recieve data for ModBusSlave ***************
void MgsModbus::MbsRun()
{
  //****************** Read from socket ***************
  EthernetClient client = MbServer.available();
  if(client.available())
  {
    delay(10);
    int i = 0;
    while(client.available())
```

```
    {
      MbsByteArray[i] = client.read();
      i++;
    }
    MbsFC = SetFC(MbsByteArray[7]);  //Byte 7 of request is FC
  }
  int Start, WordDataLength, ByteDataLength, CoilDataLength, MessageLength;
  //****************** Read Coils (1 & 2) *********************
  if(MbsFC == MB_FC_READ_COILS || MbsFC == MB_FC_READ_DISCRETE_INPUT) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    CoilDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    ByteDataLength = CoilDataLength / 8;
    if(ByteDataLength * 8 < CoilDataLength) ByteDataLength++;
    CoilDataLength = ByteDataLength * 8;
    MbsByteArray[5] = ByteDataLength + 3; //Number of bytes after this one.
    MbsByteArray[8] = ByteDataLength;     //Number of bytes after this one (or number of bytes of data).
    for(int i = 0; i < ByteDataLength ; i++)
    {
      MbsByteArray[9 + i] = 0; // To get all remaining not written bits zero
      for(int j = 0; j < 8; j++)
      {
        bitWrite(MbsByteArray[9 + i], j, GetBit(Start + i * 8 + j));
      }
    }
    MessageLength = ByteDataLength + 9;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Read Registers (3 & 4) ******************
  if(MbsFC == MB_FC_READ_REGISTERS || MbsFC == MB_FC_READ_INPUT_REGISTER) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    WordDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    ByteDataLength = WordDataLength * 2;
    MbsByteArray[5] = ByteDataLength + 3; //Number of bytes after this one.
    MbsByteArray[8] = ByteDataLength;     //Number of bytes after this one (or number of bytes of data).
```

```
    for(int i = 0; i < WordDataLength; i++)
    {
      MbsByteArray[ 9 + i * 2] = highByte(MbData[Start + i]);
      MbsByteArray[10 + i * 2] =  lowByte(MbData[Start + i]);
    }
    MessageLength = ByteDataLength + 9;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Coil (5) **********************
  if(MbsFC == MB_FC_WRITE_COIL) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    if (word(MbsByteArray[10],MbsByteArray[11]) == 0xFF00){SetBit(Start,true);}
    if (word(MbsByteArray[10],MbsByteArray[11]) == 0x0000){SetBit(Start,false);}
    MbsByteArray[5] = 2; //Number of bytes after this one.
    MessageLength = 8;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Register (6) *****************
  if(MbsFC == MB_FC_WRITE_REGISTER) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    MbData[Start] = word(MbsByteArray[10],MbsByteArray[11]);
    MbsByteArray[5] = 6; //Number of bytes after this one.
    MessageLength = 12;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Multiple Coils (15) **********************
  if(MbsFC == MB_FC_WRITE_MULTIPLE_COILS) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    CoilDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    MbsByteArray[5] = 6;
    for(int i = 0; i < CoilDataLength; i++)
    {
```

```
      SetBit(Start + i,bitRead(MbsByteArray[13 + (i/8)],i-((i/8)*8)));
    }
    MessageLength = 12;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
  //****************** Write Multiple Registers (16) ******************
  if(MbsFC == MB_FC_WRITE_MULTIPLE_REGISTERS) {
    Start = word(MbsByteArray[8],MbsByteArray[9]);
    WordDataLength = word(MbsByteArray[10],MbsByteArray[11]);
    ByteDataLength = WordDataLength * 2;
    MbsByteArray[5] = 6;
    for(int i = 0; i < WordDataLength; i++)
    {
      MbData[Start + i] =  word(MbsByteArray[ 13 + i * 2],MbsByteArray[14 + i * 2]);
    }
    MessageLength = 12;
    client.write(MbsByteArray, MessageLength);
    MbsFC = MB_FC_NONE;
  }
}


//****************** ?? ******************
MB_FC MgsModbus::SetFC(int fc)
{
  MB_FC FC;
  FC = MB_FC_NONE;
  if(fc == 1) FC = MB_FC_READ_COILS;
  if(fc == 2) FC = MB_FC_READ_DISCRETE_INPUT;
  if(fc == 3) FC = MB_FC_READ_REGISTERS;
  if(fc == 4) FC = MB_FC_READ_INPUT_REGISTER;
  if(fc == 5) FC = MB_FC_WRITE_COIL;
  if(fc == 6) FC = MB_FC_WRITE_REGISTER;
  if(fc == 15) FC = MB_FC_WRITE_MULTIPLE_COILS;
```

```
  if(fc == 16) FC = MB_FC_WRITE_MULTIPLE_REGISTERS;
  return FC;
}


word MgsModbus::GetDataLen()
{
  return MbDataLen;
}


boolean MgsModbus::GetBit(word Number)
{
  int ArrayPos = Number / 16;
  int BitPos = Number - ArrayPos * 16;
  boolean Tmp = bitRead(MbData[ArrayPos],BitPos);
  return Tmp;
}


boolean MgsModbus::SetBit(word Number,boolean Data)
{
  int ArrayPos = Number / 16;
  int BitPos = Number - ArrayPos * 16;
  boolean Overrun = ArrayPos > MbDataLen * 16; // check for data overrun
  if (!Overrun){
    bitWrite(MbData[ArrayPos],BitPos,Data);
  }
  return Overrun;
}
```