```
/////////////////// Port information //////////////////
#define baud 19200
#define timeout 1000
#define polling 10 // the scan rate
```

Make sure these parameters match the settings of the device you want to communicate with!

```
// If the packets internal retry register matches
// the set retry count then communication is stopped
// on that packet. To re-enable the packet you must
// set the "connection" variable to true.
#define retry_count 10

// used to toggle the receive/transmit pin on the driver
#define TxEnablePin 20

#define LED 9
```

Not required if using the AET RS485 Shield. Only here to toggle an LED to check that the communication is still running...

```
// This is the easiest way to create new packets
// Add as many as you want. TOTAL_NO_OF_PACKETS
// is automatically updated.
enum
{
  PACKET1,
  PACKET2,
  PACKET3,
  PACKET4,
  PACKET5,
  PACKET6,
  PACKET7,
  PACKET8,
  PACKET9,
  PACKET10,
  PACKET11,
  PACKET12,
  PACKET13,
  PACKET14,
  PACKET15,
  PACKET16,
  PACKET17,
  TOTAL_NO_OF_PACKETS // leave this last entry
};
```

You need one packet for each register that you are wanting to read or write to. This enum is used to assemble all the packets into sequence and to check the total packet count...

```
// Create an array of Packets to be configured
Packet packets[TOTAL_NO_OF_PACKETS];
```

The array of packets which will be sent out...

```
// Create a packetPointer to access each packet
// individually. This is not required you can access
// the array explicitly. E.g. packets[PACKET1].id = 2;
// This does become tedious though...
//////////////////////////HMI READS///////////////////////////
packetPointer enableDisableHMI = &packets[PACKET1];
packetPointer runFwdHMI = &packets[PACKET2];
packetPointer runRevHMI = &packets[PACKET3];
packetPointer resetHMI = &packets[PACKET4];
packetPointer eStopHMI = &packets[PACKET5];
packetPointer userSetSpeedHMI = &packets[PACKET6];

//////////////////////////HMI WRITES//////////////////////////
packetPointer runLedHMI = &packets[PACKET7];
packetPointer stopLedHMI = &packets[PACKET8];
packetPointer errorLedHMI = &packets[PACKET9];
packetPointer actualSpeedHMI = &packets[PACKET10];
packetPointer motorCurrentHMI = &packets[PACKET11];

/////////////////////////VSD READS////////////////////////////
packetPointer statusWordVSD = &packets[PACKET12];
packetPointer actualSpeedVSD = &packets[PACKET13];
packetPointer motorCurrentVSD = &packets[PACKET14];

/////////////////////////VSD WRITES//////////////////////////
packetPointer commandWordVSD = &packets[PACKET15];
packetPointer userSetSpeedVSD = &packets[PACKET16];
packetPointer clearFaultsVSD = &packets[PACKET17];
```

Each register variable which you will be using is assigned to a packet using these pointer declarations...

```
////////HMI READ VARIABLES///////////
unsigned int readEnableDisableHMI[1];
unsigned int readRunFwdHMI[1];
unsigned int readRunRevHMI[1];
unsigned int readResetHMI[1];
unsigned int readEstopHMI[1];
unsigned int readUserSetSpeedHMI[1];

////////HMI WRITE VARIABLES//////////
unsigned int writeRunLedHMI[1];
unsigned int writeStopLedHMI[1];
unsigned int writeErrorLedHMI[1];
unsigned int writeActualSpeedHMI[1];
unsigned int writeMotorCurrentHMI[1];

////////VSD READ VARIABLES///////////
unsigned int readStatusWordVSD[1];
unsigned int readActualSpeedVSD[1];
unsigned int readMotorCurrentVSD[1];

////////VSD WRITE VARIABLES//////////
unsigned int writeControlWordVSD[1];
unsigned int writeUserSetSpeedVSD[1]={0};
unsigned int writeClearFaultsVSD[1];
```

The actual variables into which the registers will be read and saved for use in your code...

```
void setup()
{
Serial.begin(19200);
 // Set modes of some pins for LED outputs
 pinMode(vccPin, OUTPUT);
 pinMode(gndPin, OUTPUT);
 digitalWrite(vccPin, HIGH);
 digitalWrite(gndPin, LOW);
 // Read all values from HMI
 modbus_construct(enableDisableHMI, 3, READ_HOLDING_REGISTERS, 50, 1, readEnableDisableHMI);
 modbus_construct(runFwdHMI, 3, READ_HOLDING_REGISTERS, 60, 1, readRunFwdHMI);
 modbus_construct(runRevHMI, 3, READ_HOLDING_REGISTERS, 70, 1, readRunRevHMI);
 modbus_construct(resetHMI, 3, READ_HOLDING_REGISTERS, 80, 1, readResetHMI);
 modbus_construct(eStopHMI, 3, READ_HOLDING_REGISTERS, 90, 1, readEstopHMI);
 modbus_construct(userSetSpeedHMI, 3, READ_HOLDING_REGISTERS, 10, 1, readUserSetSpeedHMI);
 // Write required values to HMI
 modbus_construct(runLedHMI, 3, PRESET_MULTIPLE_REGISTERS, 100, 1, writeRunLedHMI);
 modbus_construct(stopLedHMI, 3, PRESET_MULTIPLE_REGISTERS, 110, 1, writeStopLedHMI);
 modbus_construct(errorLedHMI, 3, PRESET_MULTIPLE_REGISTERS, 120, 1, writeErrorLedHMI);
 modbus_construct(actualSpeedHMI, 3, PRESET_MULTIPLE_REGISTERS, 0, 1, readActualSpeedVSD);
 modbus_construct(motorCurrentHMI, 3, PRESET_MULTIPLE_REGISTERS, 20, 1, readMotorCurrentVSD);
 // Read all values from VSD
 modbus_construct(statusWordVSD, 2, READ_HOLDING_REGISTERS, 8603, 1, readStatusWordVSD);
 modbus_construct(actualSpeedVSD, 2, READ_HOLDING_REGISTERS, 8604, 1, readActualSpeedVSD);
 modbus_construct(motorCurrentVSD, 2, READ_HOLDING_REGISTERS, 3204, 1, readMotorCurrentVSD);
 // Write required values to VSD
 modbus_construct(commandWordVSD, 2, PRESET_MULTIPLE_REGISTERS, 8601, 1, writeControlWordVSD);
 modbus_construct(userSetSpeedVSD, 2, PRESET_MULTIPLE_REGISTERS, 8602, 1, readUserSetSpeedHMI);
 modbus_construct(clearFaultsVSD, 2, PRESET_MULTIPLE_REGISTERS, 8501, 1, writeClearFaultsVSD);
 // Configure the MODBUS connection
 modbus_configure(baud, SERIAL_8E1, timeout, polling, retry_count, TxEnablePin, packets, TOTAL_NO_OF_PACKETS);
}
```

These constructs add all fields required by the modbus protocol to your packets. You need to have one construct for each packet...

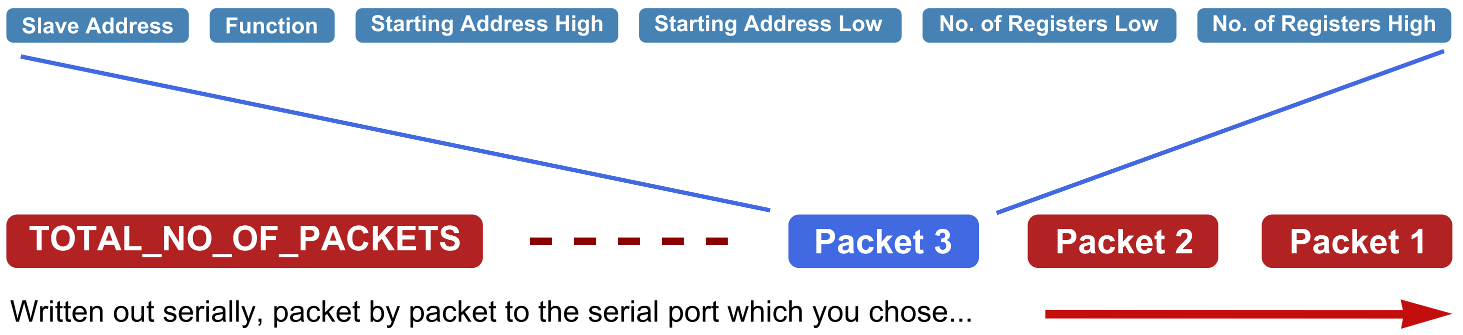
This final function call, "modbus_configure" sets up all of the connection parameters for your communication between the master (Arduino) and your slave device...

Software Required for this system setup:

1. DELTA DOPsoft for HMI screen design, programming and register and communication setup.

2. Arduino IDE with the AET SimpleModbusMater library loaded. (see https://github.com/aetcnc/RS485-Shield)

3. Sample program from the above GITHUB repository which grabs key press information from the HMI modbus registers and turns them into a "BUTTONSTATE" variable so the user can tell which button has been pressed.



Written out serially, packet by packet to the serial port which you chose...

Remember... this is **Modbus RTU ONLY** !!! The "SimpleModbusMaster" library does not yet support ASCII !!! If you try and implement Modbus ASCII with this library you will be pulling hair out for all eternity... trust us...

Make sure to check on our YouTube page for videos of the HMI DOPSoft setup. It shows how to build up the screen shown on page 2 and you will quickly learn how to go about making your own custom HMI.

www.youtube.com/aetcnc

DELTA DOP-B07E515
Prepared By: GHJ   Date: 12/12/2013

Affordable•Engineering•Technologies

Delta HMI with Arduino - Sheet 3/3
- Software Implementation Guide