How to interface with TOTUS using C#.NET and MODBUS API

by David Luca

16 Sept 2014

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Comments |
| 01 | 16 Sept 2014 | David Luca | First draft |

# Introduction

This application note describes how to use the MODBUS API to retrieve data from the Camlin Power TOTUS or INTEGO devices using Visual Studio and C#.NET. It includes details on how to configure the TOTUS for remote MODBUS access, how to connect via MODBUS/TCP or MODBUS-RTU and how to retrieve measurement and alarm results.

This document does not attempt to describe the details of the MODBUS protocol. For more detailed information, please refer to the MODBUS specifications (<http://www.modbus.org/specs.php>).

Full source code for this application note is available for download from: <https://github.com/davidlcamlin/totus_modbus>

Note that this document refers to the TOTUS product, but the instructions are equally applicable for the INTEGO product.

# Pre-requisites

* **Visual Studio** - this application note was developed and tested using Visual Studio C# Express 2013.
* **NModbus library** – this application note used [NModbus .NET 3.5 version 1.11.0.0](https://code.google.com/p/nmodbus/downloads/detail?name=NModbus_net-3.5_1.11.0.0.zip&can=2&q=). This and other versions may be downloaded from the NModbus project on google code: <https://code.google.com/p/nmodbus/>. Detailed documentation is provided in a help file included in the download zip. NModbus is released under the MIT license.

# Configure MODBUS on TOTUS

The first step is to enable the MODBUS interface on the TOTUS device. This requires access to the embedded web server on the TOTUS. If using the local direct connection available on LAN3, the TOTUS web interface is available at <http://192.168.7.2>. Otherwise your network administrator should be able to provide the necessary URL.

The TOTUS supports both serial and TCP variants of the MODBUS protocol. Using the TOTUS web interface, navigate to the Settings->MODBUS page to configure which MODBUS protocol to use. MODBUS-RTU and MODBUS-ASCII may be used over RS232 or RS485, while MODBUS/TCP may be used over TCP/IP connections (e.g. LAN or Cellular).

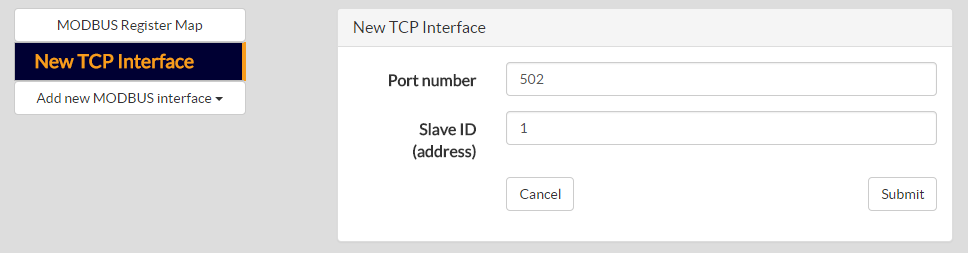
## Configure MODBUS for TCP/IP Connections

Click “Add New MODBUS Interface” and select “New TCP interface”.

For this application note, we shall use the following settings:

* Slave Address: 1
* Port: 502

Configure the MODBUS interface with the above settings and click Submit.



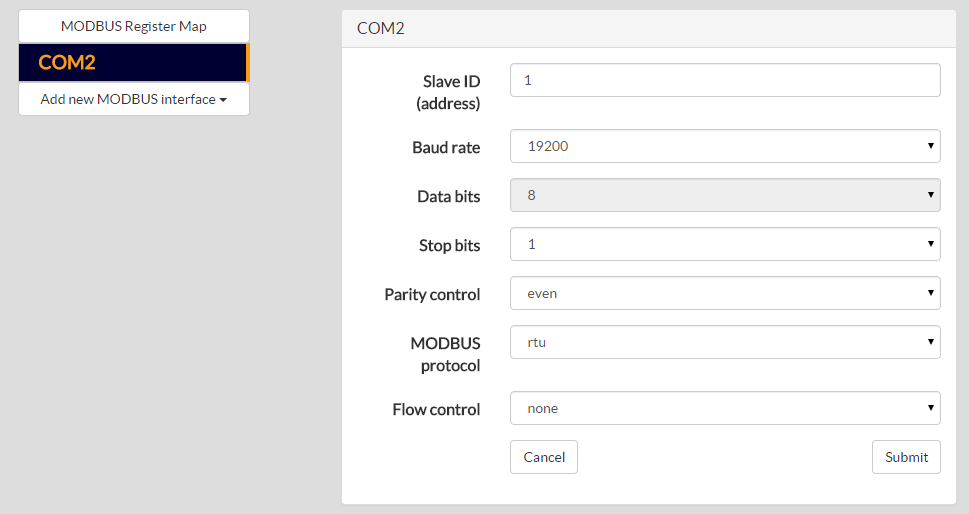
## Configure MODBUS for Serial Connections

Click “Add New MODBUS Interface” and select the desired COM port. Please note that different COM ports support different serial protocols (i.e. RS232, RS485, etc.).

For this application note, we shall use the following settings:

* Slave Address: 1
* Baud Rate: 19200
* Data bits: 8
* Stop bits: 1
* Parity: Even
* Protocol: MODBUS-RTU
* Flow control: None

Configure the MODBUS interface with the above settings and click Submit.



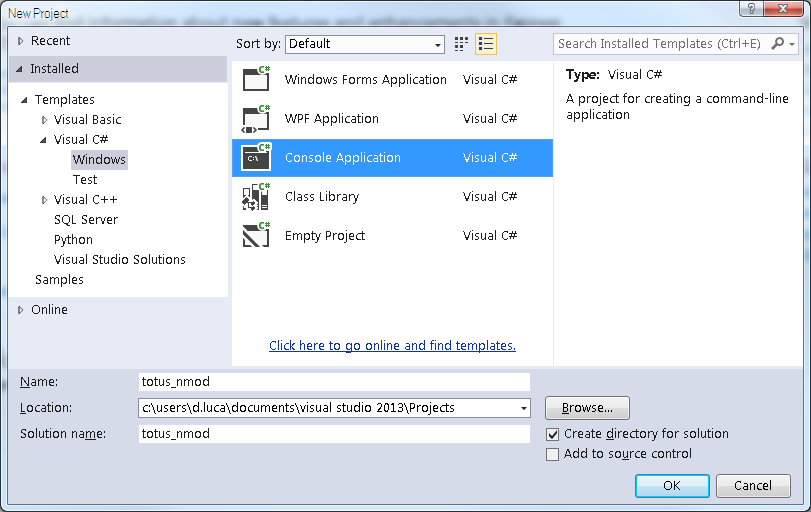
# Download TOTUS MODBUS Register Map

The TOTUS MODBUS Register Map is a document that specifies all the available registers on the TOTUS that may be read or written via MODBUS. It includes details on the address, size, format and scaling of the data.

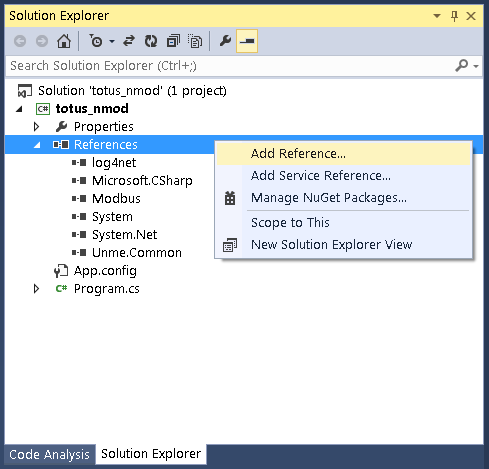
The TOTUS MODBUS Register Map may be downloaded directly from the device. Navigate to the Settings->MODBUS page and click on the link “Download Register Map”.

# Setup Visual Studio Project

Open Visual Studio and create a new Visual C# Console Application Project and name it totus\_nmod.



In Solution Explorer expand the project and right-click References line, select Add References… and browse for **Modbus.dll, Log4net.dll and Unme.Common.dll** extracted from NModbus download.



# Create a MODBUS Connection to TOTUS

In this section we shall create a connection to the TOTUS device using the NModbus library. Depending on the communications infrastructure, this connection may be via either TCP or serial.

Edit the Program.cs and add the following using directives:

using System.Net;

using System.Net.Sockets;

using System.IO.Ports;

using Modbus.Device;

Now choose whether to connect via TCP or Serial.

## TCP Connection

Add the following code to the Main function to open the TCP socket and create a MODBUS/TCP master:

TcpClient client = new TcpClient("192.168.7.2", 502); // Change for your TOTUS IP

ModbusIpMaster master = ModbusIpMaster.CreateIp(client);

master.Transport.ReadTimeout = 1000; //ms

Note that the IP address above should be modified to match the IP address of your TOTUS. The above code uses the default TCP port 502 as was configured on the TOTUS.

## Serial Connection

Add the following code to the Main function to open the serial port and create a MODBUS-RTU master:

SerialPort serialPort = new SerialPort();

serialPort.PortName = "COM5"; // Change this for your PC COM port

serialPort.BaudRate = 19200;

serialPort.DataBits = 8;

serialPort.Parity = Parity.Even;

serialPort.StopBits = StopBits.One;

serialPort.RtsEnable = false;

serialPort.Open();

ModbusSerialMaster master = ModbusSerialMaster.CreateRtu(serialPort);

master.Transport.ReadTimeout = 1000; //ms

Note that the PortName should be modified to match the local PC COM port which is connected to the TOTUS. The above code uses the serial port settings that were configured on the TOTUS.

# Read Alarms from TOTUS

Alarms are implemented as MODBUS Input Registers. These are read using the ReadInputs method on the ModbusMaster object. This method requires as parameters: the slave ID, start address and number of inputs to read. The slave ID is as was configured on the TOTUS. The start address and number of inputs is determined from the TOTUS MODBUS Register Map, downloaded earlier. The ReadInputs method returns an array of Boolean values for the requested input registers.

The following code will read the HL Alarm and HHLL Alarm states from the TOTUS:

byte slaveID = 1; //as configured on the TOTUS

ushort startAddress = 100; //see TOTUS MODBUS Register Map

ushort numInputs = 2; //read 2 alarms

//see TOTUS MODBUS Register Map

string[] totusAlarms = {

"ALARM/System/HL/State",

"ALARM/System/HHLL/State"

};

bool[] alarms = master.ReadInputs(slaveID, startAddress, numInputs);

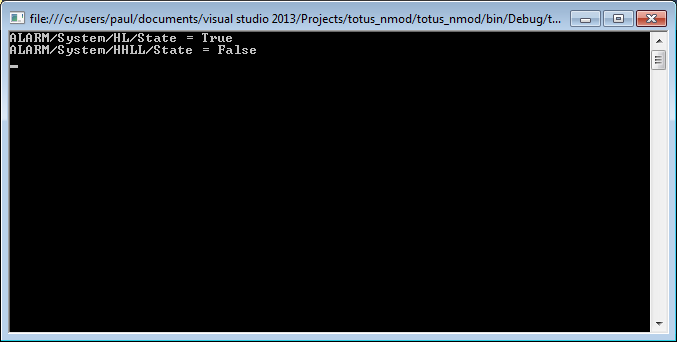
for (int i = 0; i < numInputs; i++)

{

Console.WriteLine("{0} = {1}", totusAlarms[i], alarms[i]);

}

Output:



# Read Measurement Results from TOTUS

Measurements are stored internally within the TOTUS in floating point representation. The official MODBUS specification only defines support for Boolean inputs and unsigned 16-bit integer input registers. It does not define how to represent floating point values. In the TOTUS, measurement results are presented in both scaled 16-bit integer registers and in 32-bit floating points cast into two adjacent 16-bit integer registers.

Registers values may be read using the ReadInputRegister method on the ModbusMaster object. This method requires as parameters: the slave ID, start address and number of inputs to read. The slave ID is as was configured on the TOTUS. The start address and number of inputs is determined from the TOTUS MODBUS Register Map, downloaded earlier. Note that the number of inputs must be multiplied by 2 when reading floating point representations of the measurement results since each floating point value occupies 2 adjacent 16-bit register. The ReadInputRegister method returns an array of ushort (16-bit) values. These must then be converted into floating points.

The following code will read the temperature and humidity measurements from the TOTUS using the scaled 16-bit representations:

byte slaveID = 1; //as configured on the TOTUS

ushort startAddress = 1000; //see TOTUS MODBUS Register Map

ushort numInputs = 10; //read 10 temperature values

//see TOTUS MODBUS Register Map

string[] totusTemps = {

"Thermal/AmbientTemp",

"Thermal/AmbientTemp/1hAvg",

"Thermal/AmbientHumidity",

"Thermal/AmbientHumidity/1hAvg",

"Thermal/TopOilTemp",

"Thermal/TopOilTemp/1hAvg",

"Thermal/BottomOilTemp",

"Thermal/BottomOilTemp/1hAvg",

"Thermal/TapChangerTemp",

"Thermal/TapChangerTemp/1hAvg"

};

ushort[] temps = master.ReadInputRegisters(slaveID, startAddress, numInputs);

for (int i = 0; i < numInputs; i++)

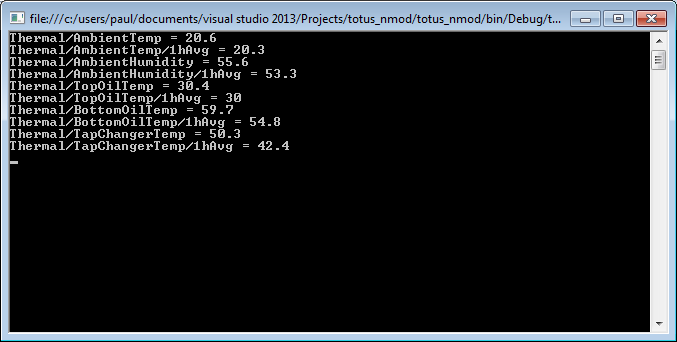
{

// results divided by 10 as specified in TOTUS MODBUS Register Map

Console.WriteLine("{0} = {1}°C", totusTemps[i], (float)temps[i] / 10);

}

Output:



The following code will read the DGA results from the TOTUS using the floating point representations in adjacent registers:

byte slaveID = 1; //as configured on the TOTUS

ushort startAddress = 2200; //see TOTUS MODBUS Register Map

ushort numInputs = 12; //read 12 DGA values

//see TOTUS MODBUS Register Map

string[] totusDGA = {

"DGA/SourceA/CH4",

"DGA/SourceA/C2H6",

"DGA/SourceA/C2H4",

"DGA/SourceA/C2H2",

"DGA/SourceA/CO",

"DGA/SourceA/CO2",

"DGA/SourceA/O2",

"DGA/SourceA/N2",

"DGA/SourceA/H2",

"DGA/SourceA/H2O",

"DGA/SourceA/TDCG",

"DGA/SourceA/THC"

};

// numInputs\*2 because we are reading 32bit floats in 2\*16bit registers

ushort[] inputsdga = master.ReadInputRegisters(slaveID,

startAddress,

(ushort)(numInputs \* 2));

for (int i = 0; i < numInputs; i++)

{

float result = Convert2Float(inputsdga[i \* 2], inputsdga[i \* 2 + 1]);

Console.WriteLine("{0} = {1} ppm", totusDGA[i], result);

}

Note that the number of registers passed to the ReadInputRegisters method was multiplied by 2. Every 2 values in the ushort array returned represents a single floating point value. They must be converted and this requires the Convert2Float method below:

static float Convert2Float(ushort high, ushort low)

{

//Convert ushort array to Float

ushort[] data = new ushort[2] { low, high }; //Big endian

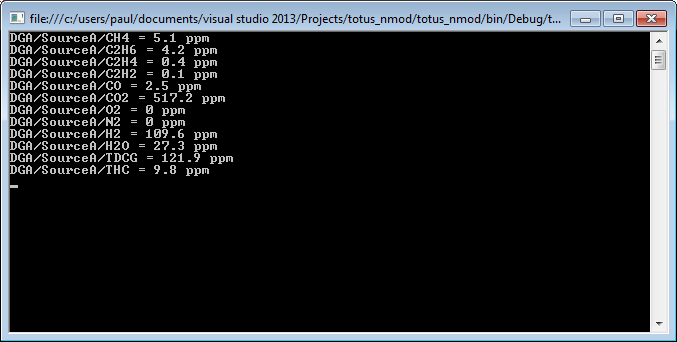
float[] floatData = new float[data.Length / 2];

Buffer.BlockCopy(data, 0, floatData, 0, data.Length \* 2);

return floatData[0];

}

Output:



# Conclusion

This application note demonstrated how to configure the TOTUS for MODBUS, how to connect and how to read a few alarms and measurements from the TOTUS. With reference to the TOTUS MODBUS Register Map, it is possible to read any value available on the TOTUS.

The above examples can be used as a starting point to build a more comprehensive, custom integration solution using the TOTUS.