TOTUS Modbus Integration Application Note

Author: David Luca

Table of Contents

[Document History 1](#_Toc399512082)

[Introduction 1](#_Toc399512083)

[Totus Modbus Register map 2](#_Toc399512084)

[Integrating in C# applications 2](#_Toc399512085)

[1. Resources 2](#_Toc399512086)

[2 Connecting to Totus 3](#_Toc399512087)

[2.1 Connecting via TCP 3](#_Toc399512088)

[2.2 Connecting via Serial port 4](#_Toc399512089)

[3 Reading values from Totus unit 6](#_Toc399512090)

[3.1 Reading Alarms 6](#_Toc399512091)

[3.2 Reading temperatures 6](#_Toc399512092)

[3.3 Reading DGA values 6](#_Toc399512093)

[Conclusions 7](#_Toc399512094)

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Comments |
| 0.1 | 16/09/2014 | David Luca | First draft |

# Introduction

The purpose of the document is to introduce customers into using available MODBUS software libraries to connect their software to TOTUS instruments.

This document assumes you can access Totus interface by typing in your browser the IP address of the unit (in this document we will use <Totus-IP>). You may have to setup VPN connection, for which you need to contact Camlin Technologies.

# Totus Modbus Register map

In TOTUS dashboard interface press **Settings** menu and select **MODBUS** menu or type in your browser http://<Totus-IP>/#modbus-settings link. There is a link “Download Register Map” for this device (or http://<Totus-IP>/modbus-map) where there is a table of MODBUS protocol description, meaning of the fields and register transfer requirements to read and/or write into the dictionary.

I.e.:

Please note that this specification is still in draft status and may change prior to formal release.

0000\_Common

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Topic** | **Address** | **Register** | **Number of Registers** | **Access** | **Format** | **Scaling** | **Offset** | **Notes** |
| UTC Time | 0 | 30001 | 2 | Read only | UINT32 |  |  | Clock time in UNIX time format; number of seconds from 1 Jan 1970. |
| ALARM/System/HL/State | 100 | 10101 | 1 | Read only | BOOLEAN |  |  | Combined system alarm for any non severe (H or L) alarm. |
| ALARM/System/HHLL/State | 101 | 10102 | 1 | Read only | BOOLEAN |  |  | Combined system alarm for any severe (HH or LL) alarm. |

# Integrating in C# applications

## 1. Resources

The code examples provided have been built using Visual Studio C# Express 2013 and latest version of NModbus ([NModbus\_net-3.5\_1.11.0.0-source.zip](https://code.google.com/p/nmodbus/downloads/detail?name=NModbus_net-3.5_1.11.0.0-source.zip&can=2&q=)) zip archive from: <https://code.google.com/p/nmodbus/downloads/list>.

Alternatively use SVN to checkout read-only sources for NModbus into an empty folder (i.e. E:\SVN):

svn checkout **http**://nmodbus.googlecode.com/svn/trunk/ nmodbus-read-only

Additional documentation for NModbus API: <http://ftp.icpdas.com/pub/cd/8000cd/napdos/modbus/nmodbus/nmodbus_api_manual_v1.2_en.pdf>

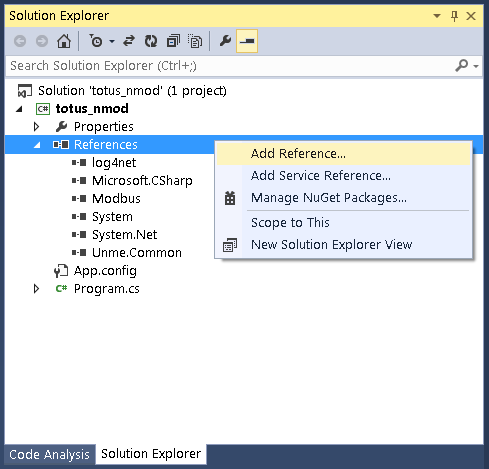
<http://www.icpdas.com/products/PAC/i-8000/modbus.htm>

https://code.google.com/p/nmodbus/

Example code is available in MySample folder inside the downloaded sources folder.

1. Create new project in Visual Studio C# Express 2013 (i.e. Totus\_nmod).

2. 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 binary archive or SVN checkout folder.

****

## 2 Connecting to Totus

### 2.1 Connecting via TCP

To access Modbus via TCP you need to press “Add New MODBUS Interface”, select “New TCP interface” then type in the form SlaveID(default 1) and the port to be used (i.e. 502, 1502) then press **Submit** button.



The following example shows how to initiate a connection to Totus unit via TCP port specified in the settings form (502):

using System.Net;//for tcp client

using System.Net.Sockets;

using Modbus.Device;//for modbus master

…..

/\*

\* Connecting via TCP

\*/

TcpClient client = new TcpClient("192.168.42.37", 502); //IP and port of the TOTUS unit

Console.WriteLine("Client connected:{0}", client.Connected.ToString());

ModbusIpMaster master = ModbusIpMaster.CreateIp(client);

master.Transport.ReadTimeout = 1000; //ms

### 2.2 Connecting via Serial port

To access Modbus via serial cable you need to press “Add New MODBUS Interface” and select the Serial0, 1 or 2, specify on the form Baudrate, SlaveID (usually 1), data and stop bits, parity control, MODBUS protocol (ASCII/RTU) and flow control (usually “none” for RS232 or “hardware” for RS485 to enable direction via RTS line). For this application note we’ll be using 115200bps, 8 data bits, no parity, 1 stop bit then press **Submit** button.



The following example shows how to connect to Totus unit via Serial port:

using Modbus.Device;//for modbus master

using System.IO.Ports;//for access to PC port

…..

/\*

\* Connecting via Serial

\*/

SerialPort serialPort = new SerialPort(); //Create a new SerialPort object.

serialPort.PortName = "COM1"; //PC port

serialPort.BaudRate = 115200; //baud rate

serialPort.DataBits = 8;

serialPort.Parity = Parity.None;

serialPort.StopBits = StopBits.One;

serialPort.RtsEnable = false; //false for RS232, true for RS485

serialPort.Open();

ModbusSerialMaster master = ModbusSerialMaster.CreateRtu(serialPort);//or .CreateAscii(serialPort)

master.Transport.ReadTimeout = 1000;//ms

The settings for baudrate, data and stop bits need to match the configured settings on Totus Settings->MODBUS page. Depending on ASCII or RTU selection use ModbusSerialMaster.CreateAscii or ModbusSerialMaster.CreateRtu functions in your code.

## 3 Reading values from Totus unit

Reading is done via ReadInputs function from library that returns array of values depending on the type of the dictionary:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

…..

### 3.1 Reading Alarms

//read alarms

byte slaveID = 1;

ushort numInputs = 2;

ushort startAddress = 100;

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

string[] totusAlarms = {

"ALARM/System/HL/State",

"ALARM/System/HHLL/State"

};

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

{

Console.WriteLine("Alarm {0} {1} = {2}", startAddress + i, totusAlarms[i], alarms[i] ? 1 : 0);

}

### 3.2 Reading temperatures

//read int16 temperatures

ushort numInputs = 10;

byte slaveID = 1;

ushort startAddress = 1000; //select address from Totus Modbus table

ushort[] temps = master.ReadInputRegisters(slaveID, startAddress, numInputs);//\*2 because we are reading 2byte unsigned short that needs converted to 4 byte floats

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

{

Console.WriteLine("Temp16 {0} {1} = {2}°C", startAddress + i, totusTemps[i], (float)temps[i] / 10); // divide by 10 as specified in Scaling column

}

### 3.3 Reading DGA values

//read DGA float32 gases

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"

};

byte slaveID = 1;

ushort numInputs = 12;

ushort startAddress = 2200; //select address from Totus Modbus table

ushort[] inputsdga = master.ReadInputRegisters(slaveID, startAddress, (ushort)(numInputs \* 2));//\*2 because we are reading 2byte unsigned short that needs converted to 4 byte floats

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

{

Console.WriteLine("Float32 {0} {1} = {2} ppm", startAddress + i \* 2, totusDGA[i], Convert2Float(inputsdga[i \* 2], inputsdga[i \* 2 + 1]));

}

The floating point value are read as 16bit values and assembled in a float using Convert2Float utility function:

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];

}

# Conclusions

This document exemplified Modbus interfacing using Open-Source libraries for various programming platforms which allows customers to implement their own Human Machine Interface to extract information from Totus units.

Example output (values with 3276.7 are for not connected sensors):

****