How to interface with TOTUS using JAVA and MODBUS API

Author: David Luca

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Comments |
| 0.1 | 16/09/2014 | 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 NetBeans and JAVA. 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

* **Netbeans IDE**
* **Oracle Java 8 SDK**
* **JaMod sources** from <http://sourceforge.net/projects/jamod/files/jamod/1.2/>

For serial RS485/RS232 you also either:

* **Java Communications API 3.0** (**Javax)** from <http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-misc-419423.html#java_comm_api-30u1> if building for Solaris/Linux or for Windows you can use <http://smslib.org/download/> that provides links to different platforms for serial communication libraries.
* **RXTX (alternative to Javax)** from Mfizz Inc.

The project Totus\_jamod described below includes the Jamod 1.2rc1 built against RXTX library since Javax cannot be used because it doesn’t support x64 bit Windows platforms. From <http://mfizz.com/oss/rxtx-for-java> download <https://bitbucket.org/jlauer/mfz-cdn/downloads/mfz-rxtx-2.2-20081207-win-x64.zip> which contains RXTXcomm.jar that needs copied into jamod-1.2rc1-src\lib folder (Jamod unzipped archive).

# 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).

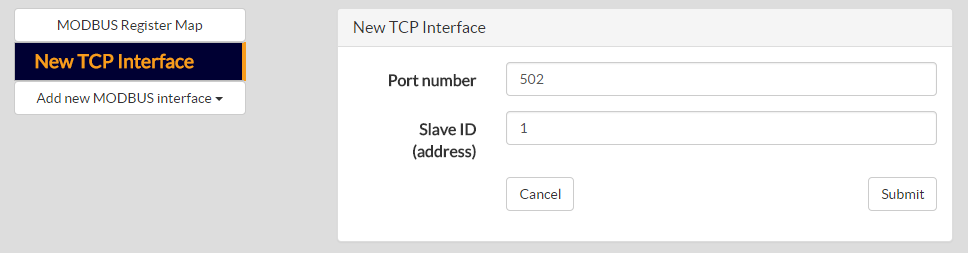
## Configure MODBUS for TCP/IP Connections

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.



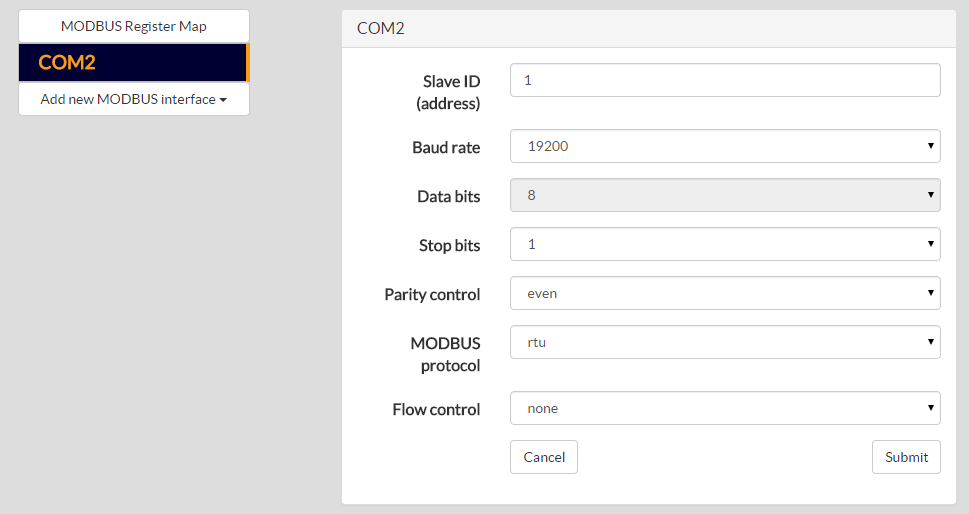
## Configure MODBUS for Serial Connections

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.

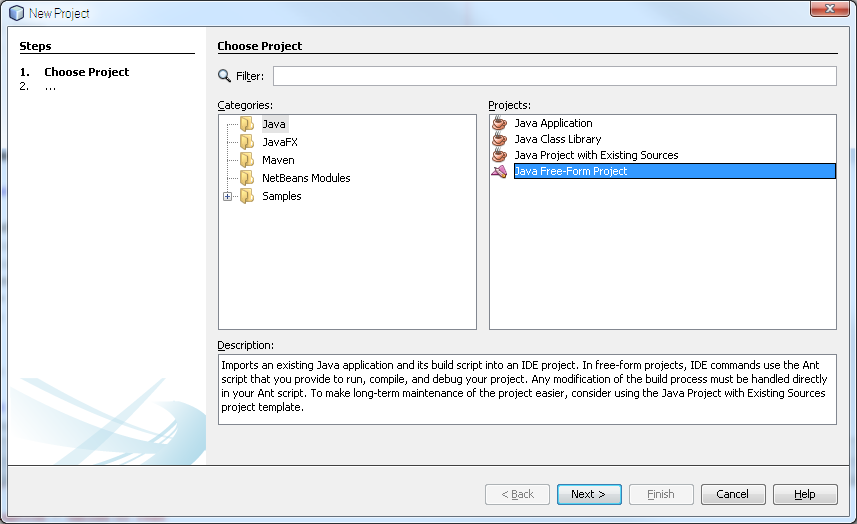
The TOTUS MODBUS Register Map may be downloaded directly from the device. Navigate to the Settings->MODBUS page and click on the link “Download Register Map”.

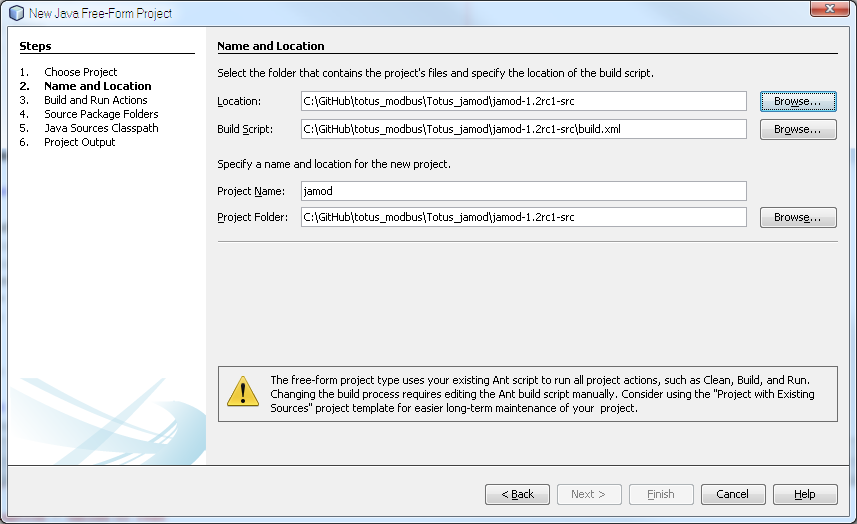
# Setup Jamod Library Project

If you need to build Jamod to use RXTX using NetBeans 8 continue reading this section, otherwise skip to next section.

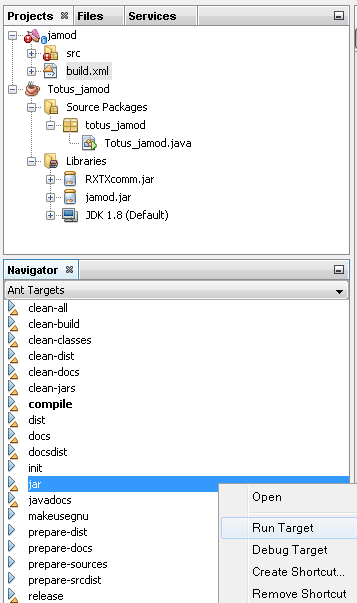
Extract the sources to an empty folder, start NetBeans IDE, select File -> New Project menu. In **Categories** pick **Java** and then in **Projects** select **Java Free-Form Project** and hit **Next** button. Configure the project by selecting the location of the project (jamod-1.2rc1-src/build.xml) and then associated targets in the build.xml will appear as actions in the IDE. Click Finish.

Edit jamod-1.2rc1-src/build.properties file to enable build.serial.gnu=true for building against RXTX (Jamod as directly downloaded from website by default uses false so it’s already built for Java Communications API through comm.jar).



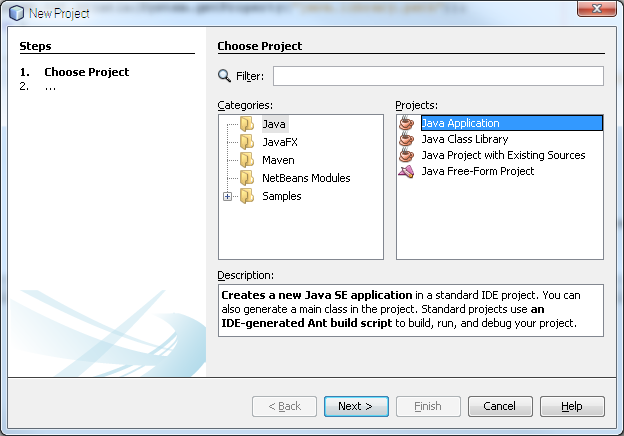


Click on build.xml and then right click on **“compile”** target, click on Run Target to compile classes, then click on **“jar”** and Run Target, which builds jamod.jar.

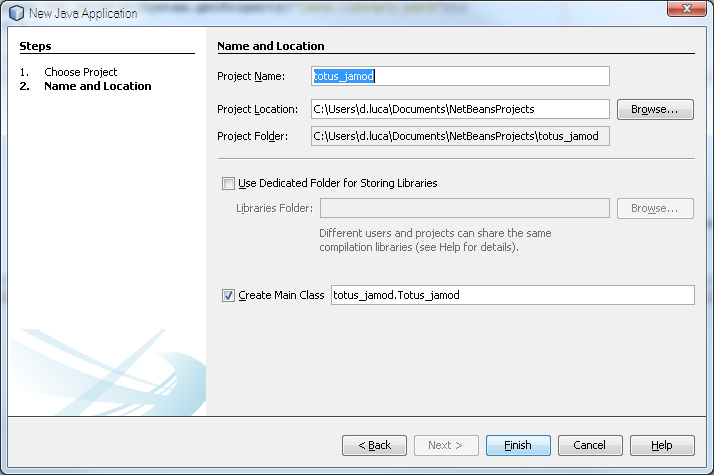


# Setup Netbeans IDE Project

Start Netbeans IDE, access File->New Project menu, select in **Categories: Java** and in **Projects:** **Java Aplication**, press Next.

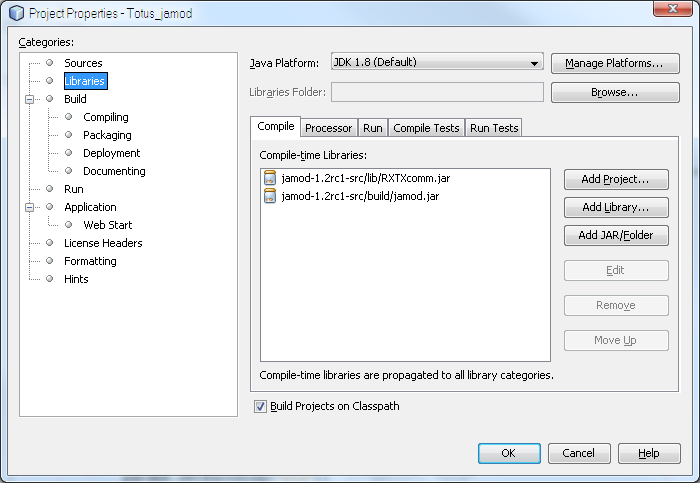


Type the name of the project (e.g. totus\_jamod) and press Finish button.



Right-click the project name (totus\_jamod), select Properties and click in the category tree “Libraries”. Click on **Add JAR/**Folder button and specify the path to Jamod library (either jamod-1.2-SNAPSHOT.jar from project website or if you need serial RXTX support in Windows use jamod-1.2rc1-src/build/jamod.jar previously rebuilt, see “Setup Jamod project”).

If you intend to use serial communication additionally include comm.jar (Java Communications API for Linux/Solaris) or jamod-1.2rc1-src/lib/RXTXcomm.jar (Windows) to project library:



# Create a MODBUS Connection to TOTUS

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

Edit Totus\_jamod.java file and add the required imports:

package totus\_jamod;

import java.io.\*;

import net.wimpi.modbus.\*;

import net.wimpi.modbus.msg.\*;

import net.wimpi.modbus.io.\*;

import net.wimpi.modbus.net.\*;

import net.wimpi.modbus.util.\*;

import java.nio.\*;

import java.net.\*;

### Connecting via TCP

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

TCPMasterConnection con =

new TCPMasterConnection( InetAddress.getByName( "192.168.42.37" ));

con.setPort(502); //port as configured on the unit

con.connect(); //connect to unit

ModbusTCPTransaction trans = new ModbusTCPTransaction(con);

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.

### Connecting via Serial

Note: Depending on your Windows platforms you need to copy the 32bit/64bit rxtxSerial.dll file in Totus\_jamod application folder or in your Windows\System32 folder.

Additional imports required to be added to your Totus\_jamod.java file:

import gnu.io.\*; //for RXTX library

//import javax.comm.\*; //for Java Communications API;

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

SerialParameters params = new SerialParameters();

params.setPortName("COM1");//PC COM port

params.setBaudRate(19200);//baudrate set in Totus unit

params.setDatabits(SerialPort.DATABITS\_8);

params.setParity(SerialPort.PARITY\_EVEN);

params.setStopbits(SerialPort.STOPBITS\_1);

params.setEncoding("rtu"); //"ascii", "rtu"

params.setEcho(false);

//params.setReceiveTimeout(3000);//not available in RXTX

params.setFlowControlIn(SerialPort.FLOWCONTROL\_NONE);//FLOWCONTROL\_NONE for RS232, FLOWCONTROL\_RTSCTS\_IN for RS485

params.setFlowControlOut(SerialPort.FLOWCONTROL\_NONE); ////FLOWCONTROL\_NONE for RS232, FLOWCONTROL\_RTSCTS\_OUT for RS485

SerialConnection con = new SerialConnection(params);

ModbusSerialTransaction trans = new ModbusSerialTransaction(con);

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 ReadInputDiscretesRequest object request on the ModbusTCPTransaction /ModbusSerialTransaction transaction object when calling its execute() and getResponse() methods. This object requires as parameters: 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 ReadInputDiscretesResponse object returned contains 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:

int startAddress = 100; //register map

int numInputs = 2;

ReadInputDiscretesRequest req = new ReadInputDiscretesRequest(startAddress, numInputs);

req.setUnitID(1); //slave ID of the unit

trans.setRequest(req);

trans.execute();

ReadInputDiscretesResponse res = (ReadInputDiscretesResponse)trans.getResponse();

String totusAlarms[] = {

"ALARM/System/HL/State",

"ALARM/System/HHLL/State"

};

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

{

boolean bit = res.getDiscretes().getBit(i);

System.out.println(totusAlarms[i] + " = " + bit);

}

# 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 ReadInputRegistersRequest method on the ModbusTCPTransaction /ModbusSerialTransaction transaction object when calling its execute() and getResponse() methods. This object 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 ReadInputRegisterResponse object returns an array of unsigned short (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:

int startAddress = 1000;

int numInputs = 10;

ReadInputRegistersRequest req = new ReadInputRegistersRequest(startAddress, numInputs);

req.setUnitID(1);

trans.setRequest(req);

trans.execute();

ReadInputRegistersResponse res = (ReadInputRegistersResponse)trans.getResponse();

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"

};

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

{

float temp = res.getRegister(i).getValue();

System.out.println(totusTemps[i] + " = " + temp / 10.0 + "°C");

}

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

int startAddress = 2200;

int numInputs = 12;

ReadInputRegistersRequest req = new ReadInputRegistersRequest(startAddress, numInputs \* 2);//float=2\*ints16

req.setUnitID(1);

trans.setRequest(req);

trans.execute();

ReadInputRegistersResponse res = (ReadInputRegistersResponse)trans.getResponse();

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"

};

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

{

float ppm = Convert2Float(res.getRegister(i \* 2).toBytes(), res.getRegister((i \* 2) + 1).toBytes());

System.out.println(totusDGA[i] + " = " + ppm + " ppm");

}

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

public static float Convert2Float(byte[] a, byte[] b)

{

ByteBuffer bbuffer = ByteBuffer.allocate(a.length + b.length);

bbuffer.put(a);

bbuffer.put(b);

bbuffer.compact(); // no need if backing array is sized appropriately to begin with

float result = ByteBuffer.wrap(bbuffer.array()).order(ByteOrder.BIG\_ENDIAN).getFloat();

return result;

}

# Conclusions

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.