How to interface with TOTUS using Python and MODBUS API

Author: David Luca

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Comments |
| 0.1 | 29/09/2014 | 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 LiClipse IDE and Python. 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

* **LiClipse IDE**
* **Python 2.7**
* **Pymodbus** (see below how to install it).
* **Setuptools** (<https://pypi.python.org/packages/source/s/setuptools/setuptools-6.0.1.zip#md5=f35cd3a424145c68c235dcb7aef89c48>) then extract into an empty folder, open a command prompt into that folder and type: python setup.py install
* **Pyserial** library: <https://pypi.python.org/pypi/pyserial> ([pyserial-2.7.win32.exe](https://pypi.python.org/packages/any/p/pyserial/pyserial-2.7.win32.exe#md5=21555387937eeb79126cde25abee4b35))
* If building under Windows also install **Microsoft Visual C++ Compiler for Python 2.7** that contains system headers necessary for producing binary wheels for Python 2.7 packages from <http://aka.ms/vcpython27>

Download <https://code.google.com/p/pymodbus/downloads/detail?name=pymodbus-0.9.0.zip> and extract it into an empty folder then open a command prompt into that folder and type:

python setup.py install

Alternatively use SVN to checkout latest version of the library:

svn checkout http://pymodbus.googlecode.com/svn/trunk/ pymodbus-read-only  
cd pymodbus-read-only  
python setup.py install

Or from GitHub:

git clone <https://github.com/bashwork/pymodbus.git>

cd pymodbus

python setup.py install

# 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).

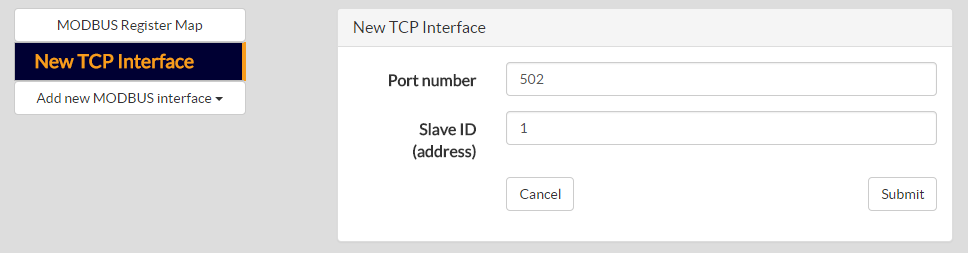
## Configure MODBUS for TCP/IP Connections

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.



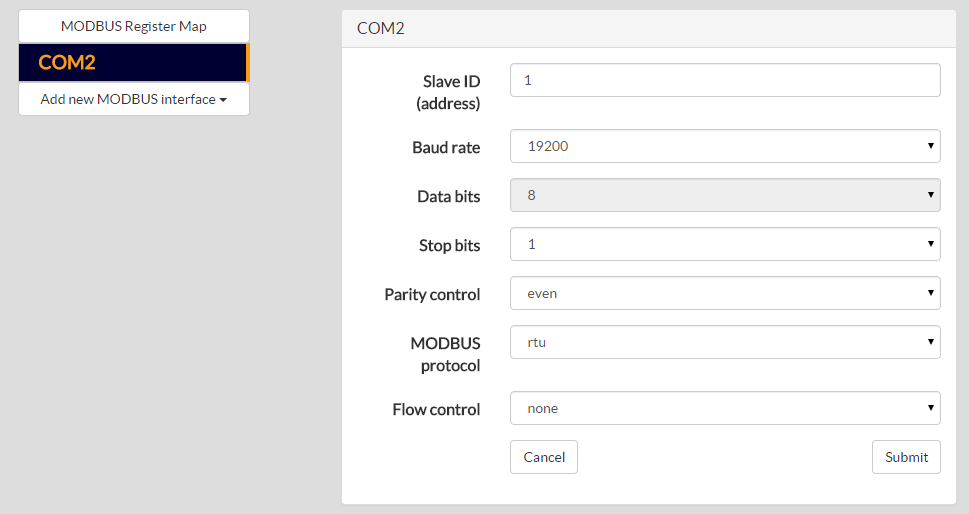
## Configure MODBUS for Serial Connections

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.



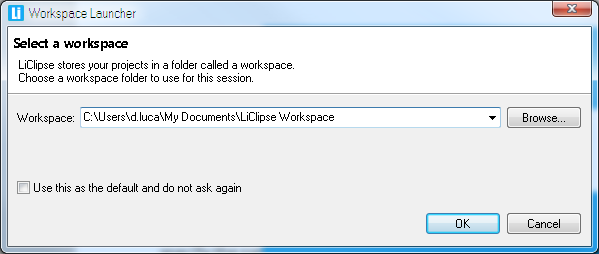
# Download TOTUS MODBUS Register Map

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.

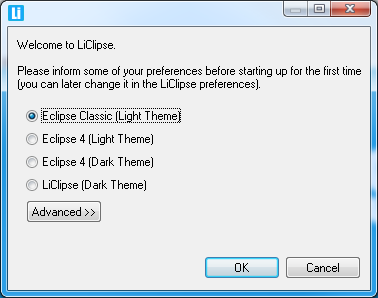
The TOTUS MODBUS Register Map may be downloaded directly from the device. Navigate to the Settings->MODBUS page and click on the link “Download Register Map”.

# Setup LiClipse Environment

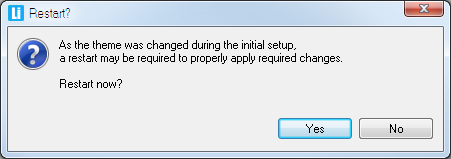
Start LiClipse and press Browse button to select the path to an empty folder then press Ok, this creates an empty workspace.



Select Eclipse Classic (Light Theme) in “Welcome to LiClipse” theme selection box and press **Ok**.



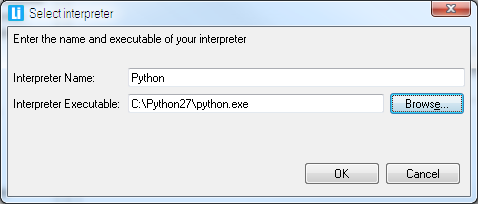
LiClipse will ask to be restarted so press **Yes** button to confirm.



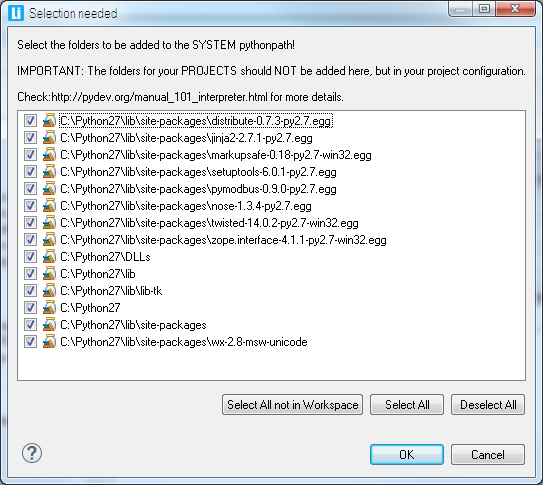
After restart, LiClipse will ask again to select the workspace and you can press **Ok** since it will have by default in Workspace edit box the path to folder selected at start-up.

Go to Window->Preferences menu, select PyDev->Interpreters->Python Interpreters in the left tree.

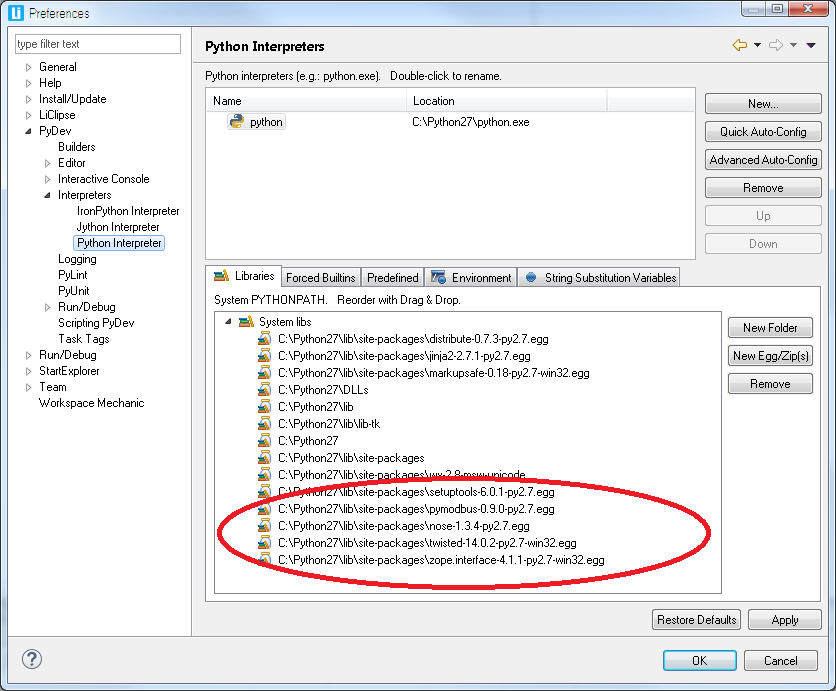
Press **Quick Auto-Config** button which will try to populate for you the path to Python 2.7 interpreter. If this cannot be found, press **New** button, type a name for interpreter (e.g. Python) and press **Browse** button to specify the path to the interpreter:



The pre-requisites should have been installed by now as specified on their download pages instructions and the environment should display the packages found. Press **Ok** button.



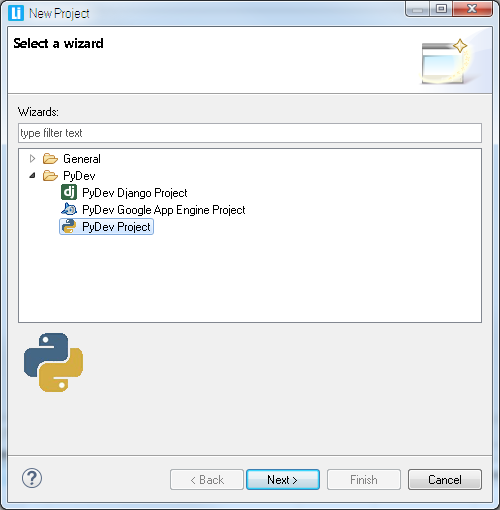
In Libraries the paths to Setuptools and PyModbus packages should be populated as below image and press **Ok** button:



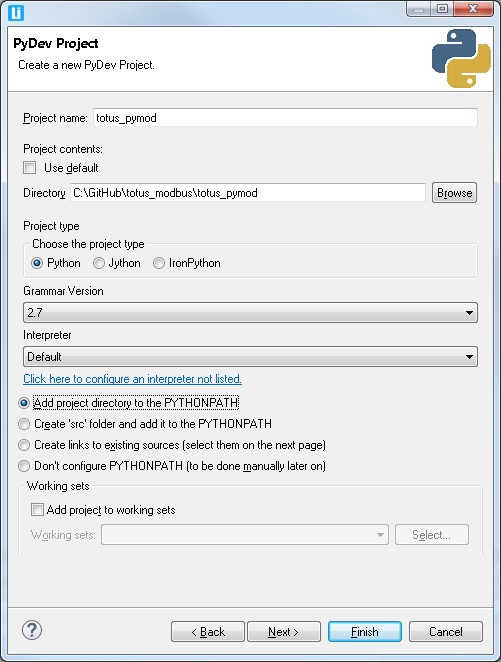
# Setup LiClipse Project

If you downloaded the examples from Git you can import the project using File->Import menu and specify the path to the project folder then skip rest of this section, otherwise continue reading.

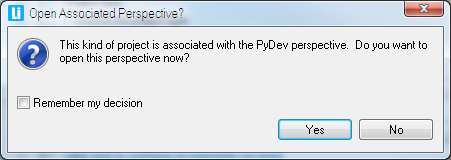
Create a new project using File->New->Project then select General->PyDev->PyDev Project in wizard window, press **Next** button.



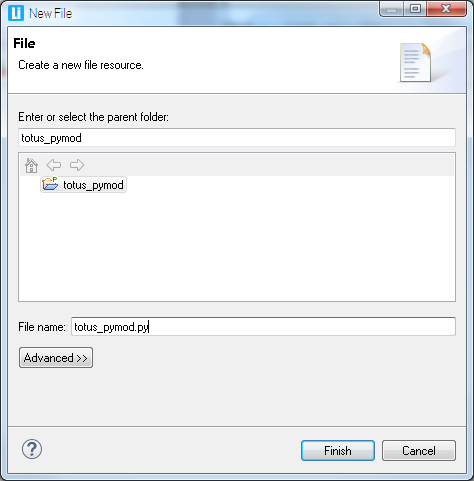
Type the project name (e.g. totus\_pymod) and select an empty folder for project location then press **Finish**.



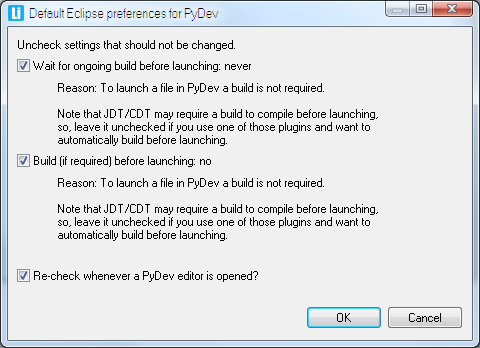
Press **Yes** button to associating the project with PyDev perspective:



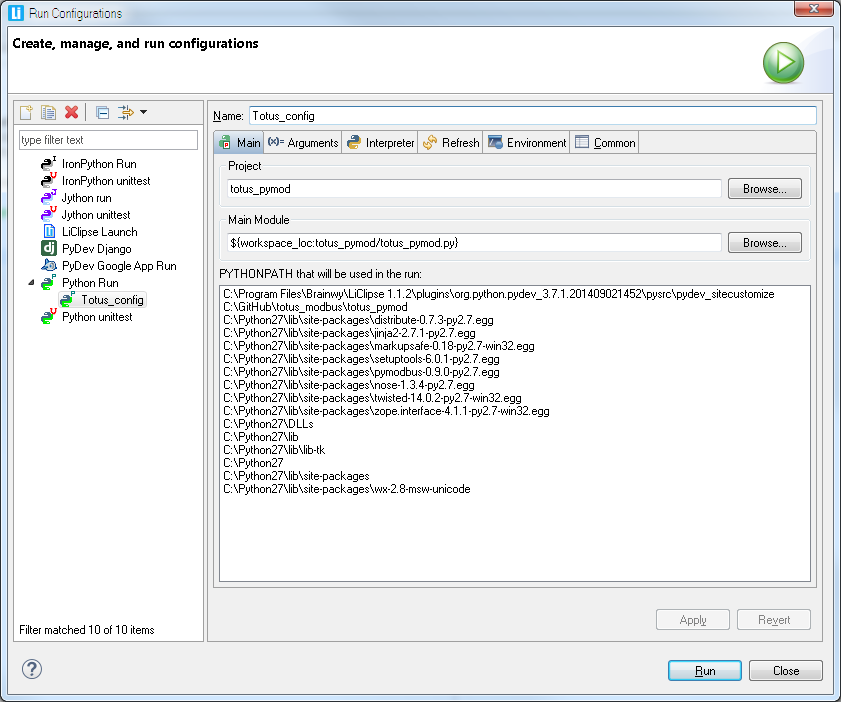
Add a new file to the project by using File->New menu, select File wizard and specify “totus\_pymod.py” and press **Finish** button.



Press **Ok** button to dialog for setting default preferences for PyDev:



Press Run button on the toolbar  and right-click on PyDev Run to add e new configuration and rename “New\_configuration” to “Totus\_config”. Press **Browse** button and select totus\_pymod project. Also in Main Module press **Browse** button and select totus\_pymod.py as start-up module project. Press **Apply** button then **Close**:



# Create a MODBUS Connection to TOTUS

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

Edit the totus\_pymode.py file and add the following import directives:

import sys, traceback

import struct

from pymodbus.constants import Endian

from pymodbus.client.sync import ModbusTcpClient as ModbusClient

Now choose whether to connect via TCP or Serial.

## TCP Connection

Use the following code to open the TCP socket and create a MODBUS/TCP master:

client = ModbusClient(*'192.168.46.83', port=502*)

client.connect()

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

For serial connection you need to replace ModbusTcpClient with ModbusSerialClient in the imports section:

from pymodbus.client.sync import ModbusSerialClient as ModbusClient

Use the following code to open the serial port and create a MODBUS-RTU master:

client = ModbusClient(method=*'rtu'*, port=*'COM5'*, timeout=1, bytesize = 8, baudrate = 19200, stopbits = 1, parity = *'E'*) # parity can be 'None', 'E'ven, 'O'dd

client.connect()

Note that the port 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 read\_input\_registers method on the ModbusClient object. This method requires as parameters: start address, number of inputs to read and the slave ID. 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 read\_input\_registers method returns an array of values for the requested input registers.

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

# read alarms

totusAlarms = [

*"ALARM/System/HL/State"*,

*"ALARM/System/HHLL/State"*

]

numInputs = 2

startAddress = 100

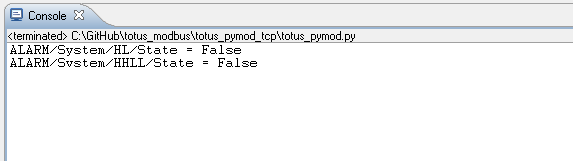
slaveID = 1

result = client.read\_discrete\_inputs(startAddress, numInputs, slaveID)

for i in range(0, len(totusAlarms)):

print totusAlarms[i] + *" = "* + str(result.getBit(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 read\_input\_registers method on the ModbusClient object. This method requires as parameters: start address, number of inputs to read and the slave ID. 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 read\_input\_registers method returns an array of registers each containing 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:

# read multiple int16 values

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"*

]

numInputs = 10

startAddress = 1000

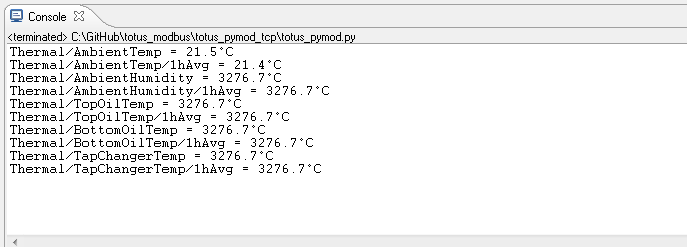
slaveID = 1

result = client.read\_input\_registers(startAddress, numInputs, slaveID)

for i in range(0, len(totusTemps)):

print totusTemps[i] + *" = "* + str(result.getRegister(i)/10.0) + *"\xb0C"*

# scaling is 10

Output (for not-connected temperature sensors displayed value is 3276.7):

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

# read DGA float32 gases

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 = 12

startAddress = 2200

slaveID = 1

result = client.read\_input\_registers(startAddress, numInputs \* 2, slaveID)

for i in range(0, len(totusDGA)):

val = Convert2Float(result.getRegister(i\*2), result.getRegister((i\*2) + 1))

print totusDGA[i] + *" = "* + str(val) + *" ppm"*

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

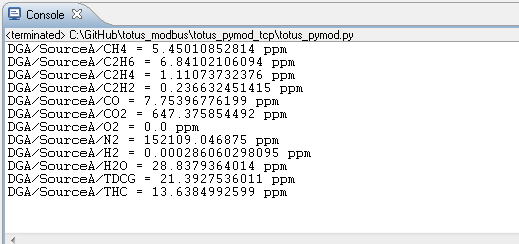
def **Convert2Float**(a, b):

raw = struct.pack(*'>HH'*, a, b)

value = struct.unpack(*'>f'*, raw)[0]

return value

Output:



# 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.