micronor
a division of PHOTON CONTROL

Application Note AN129

How to interface to the Weidmann FOTEMP-T2 Signal Conditioner

1. Objective

The Weidmann FOTEMP-T2 (formerly InsuLogix T2) unit offers versatile interface capabilities with four available interface options. This application note describes the interfaces, the software support available, how to establish communication with each interface option and sample scripts for logging temperature.

2. Interfaces Available

Each FOTEMP-T2 unit offers a different set of interface options depending on the configuration. The specific configuration is determined at the time when the unit is purchased.

There are a total of four interface options available, two of which are built-in to each unit. All units have a USB interface with ASCII communication protocol and an SD card for standalone data logging. The third interface option is RS485 which provides communication via ModbusRTU; the final interface option is ethernet which provides communication via ModbusTCP.

Units will have either RS485 or ethernet but not both. See the ordering key below along with two quick ship configurations.

Part Number:	Description:
FOTEMP-T2-X-Y-Z	Fiber Optic DIN Rail Mount Signal Conditioner where **X=No. of Channels:** up to 16 channels **Y=Interface:** P1=USB+RS485/ModbusRTU, P2=USB+Ethernet/ModbusTCP **Z=Calibration Option:** A=Basic=-40°C to +200°C, B=HighTemp=-40°C to +300°C, C=Extended=-200°C to +300°C
FOTEMP-T2-8-P1-A	8 Channels, USB+RS485/ModbusRTU, Basic Calibration=-40°C to +200°C
FOTEMP-T2-8-P2-A	8 Channels, USB+Ethernet/ModbusTCP, Basic Calibration=-40°C to +200°C

3. Interface Communication

The following interfaces are covered below:

- USB – Serial ASCII
- SD Card
- RS485 – ModbusRTU
- Ethernet – ModbusTCP

3.1. USB – Serial ASCII

The USB interface uses a Virtual Com Port (VCP) to communicate. Please refer to Micronor AN123 for a detailed description of the ASCII protocol and the list of commands.

The USB connection also provides compatibility with the FOTEMP-Assistant software running on a PC.

3.2. SD Card

All FOTEMP-T2 units come with an SD card for standalone data logging. Using Modbus communication, the data logging can be turned on or off, the sampling rate can be set and the SD data can be deleted. See pages 9 and 11 of the InsuLogix T2 Modbus Protocol Manual for more details.

Using ASCII communication, the sampling rate can be set and it is possible to read and delete the SD data. See pages 11-15 of Micronor AN123 for more details.

3.3. RS485 – ModbusRTU

ModbusRTU serial parameters settings:

Protocol	ModbusRTU
Interface	RS485 (two-wire+GND)
Baud Rate	19200
Data-bits	8
Parity	even
Stop-bits	1
End of Line	none
Slave Address	1 (default)

Table 1. ModbusRTU Protocol

To communicate with the FOTEMP-T2 unit via RS485, a serial interface from a PC or a USB-to-RS485 adapter is required. There are many options for USB-to-RS485 adaptors and most all have a male D-Sub9 connector. The connection to the FOTEMP-T2 unit is a three-wire connection as seen in *Figure 1*.

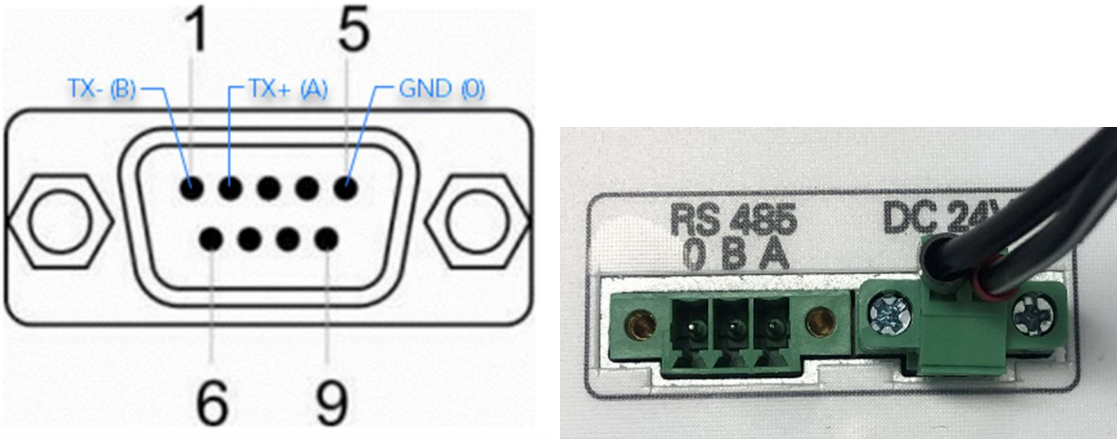

Figure 1. D-Sub9 Female Connection (left) and FOTEMP-T2 Terminal (right)

D-Sub9 Female (there are various pin designators used in the industry)	FOTEMP-T2 Terminal Connector
(1) TX- | D- | B	B
(2) TX+ | D+ | A	A
(5) GND	0

Table 2. RS485 Connection

Once connected, the device will appear in the Device Manager on the PC under "Ports (COM & LPT)" as seen in *Figure 2*, although the device description may vary depending on the USB-to-RS485 adapter used.

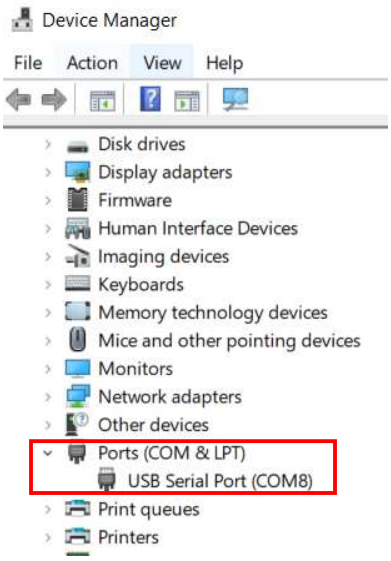

Figure 2. PC Device Manager

Once properly connected, communicating using ModbusRTU is achieved using a Python application, an IDE or a Modbus Master Simulator. This example uses the Python application provided with the FOTEMP-T2 unit. Python can be downloaded from https://www.python.org/downloads/release/python-276/.

For experimenting and writing applications, sample Python scripts are available on the FOTEMP zip drive inside the "Software\Scripts\Single_Slave_Device" folder and the Modbus registry is found in the InsuLogix T2 Modbus Protocol Manual.

This example will demonstrate how to change the address of the FOTEMP-T2 unit. Begin by opening the "Software\Scripts\Single_Slave_Device" folder. Double click on "Set_Address.py" and *Figure 3* will appear. Enter the port number from *Figure 2* and press enter to connect with the unit. Finally, enter the desired address (limited to 1-255) and press enter. The Modbus ID of the FOTEMP-T2 unit will now read the entered address (255 in this case).

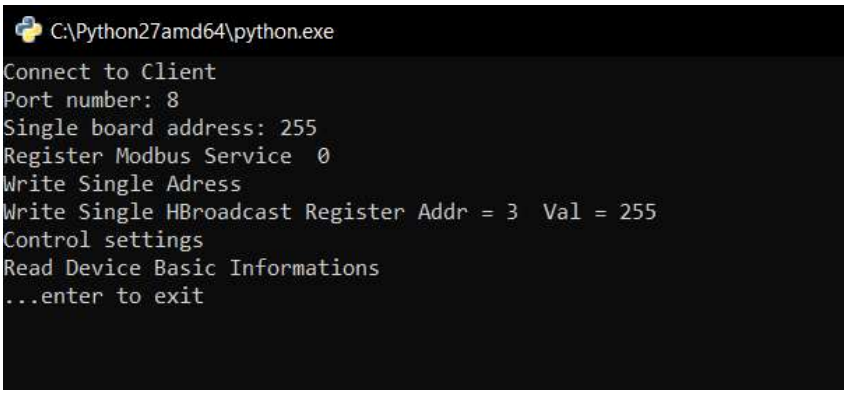

Figure 3. Set_Address.py Executable with ModbusRTU

3.4. Ethernet – ModbusTCP

ModbusTCP is a TCP packet that embeds ModbusRTU, thus serial parameters still must be configured so that the FOTEMP-T2 unit can communicate properly. For more information about Modbus communication, see Micronor AN119.

ModbusTCP serial parameters settings:

Protocol	ModbusTCP
Interface	Ethernet
Baud Rate	19200
Data-bits	8
Parity	even
Stop-bits	1
End of Line	none
Slave Address	1 (default)

Table 3. ModbusTCP Protocol

To communicate with the FOTEMP-T2 unit via ethernet, an ethernet connection or a USB-to-ethernet adapter is required. Once connected, the device will appear in the Device Manager under "Network adapters" as seen in *Figure 4*, although the device description may vary.

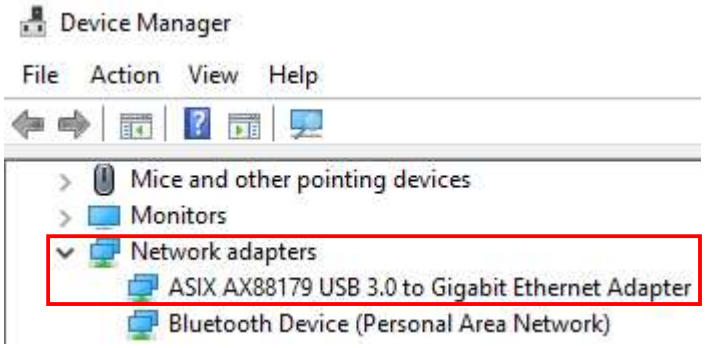

Figure 4. PC Device Manager

Communicating via ModbusTCP is very similar to ModbusRTU, so follow the steps in Section 3.3 to download Python. However, there are two main differences when using ModbusTCP. First, it uses the scripts in the "Software\Scripts\Single_Client_Device" folder. Second, rather than using a COM port, ModbusTCP uses the IP address of the FOTEMP-T2 unit. The IP address can be found by using the Lantronix DeviceInstaller software found at https://www.lantronix.com/products/deviceinstaller/.

Once the IP address is known, the code in modbuslib.py must be changed to match ModbusTCP protocol. Open the "Software\Scripts\Single_Client_Device" folder and then open modbuslib.py in your preferred coding environment (e.g., Visual Studio). Change the ModbusClient to a ModbusTcpClient as seen in *Figure 5*. Save the file.

modbuslib.py

```
#!/usr/bin/env python
# Author: Sven Geissler
# Created: 2016/09/13
# Company: Optocon AG,
# Script's name = opto_modbus_client_helpers
# Helper tools for testing Modbus implementation on Optocon Fotemp devices

#from pymodbus.client.sync import ModbusSerialClient as ModbusClient
from pymodbus.client.sync import ModbusTcpClient as ModbusClient


from pymodbus.constants import DeviceInformation
```

Figure 5. Updated modbuslib.py for ModbusTCP

This example will demonstrate how to change the address of the FOTEMP-T2 unit. Begin by reopening the "Software\Scripts\Single_Client_Device" folder. Double click on "Set_Address.py" and *Figure 6* will appear. Enter the IP address of the FOTEMP-T2 unit and press enter to connect with the unit. Then, enter the desired address (limited to 1-255) and press enter. The Modbus ID of the FOTEMP-T2 unit will now read the entered address (1 in this case).

```
C:\Python27amd64\python.exe
Connect to Client
IP Address:
Single board address: 1
Register Modbus Service  0
Write Single Adress
Write Single HBroadcast Register Addr = 3  Val = 1
'ModbusTcpClient' object has no attribute '_in_waiting'
...enter to exit
```

Figure 6. Set_Address.py Executable with ModbusTCP

4. Application Programming with FOTEMP Devices

For logging temperatures using a FOTEMP-T2 device, begin by downloading Python from https://www.python.org/downloads/release/python-276/. The following sections demonstrate how to display the temperature measurements using a Python script. These scripts can also be found on the FOTEMP zip drive in the "Software\Application Programming\Python_Scripts" folder. *Figure 7* and *Figure 8* display the output of each script.

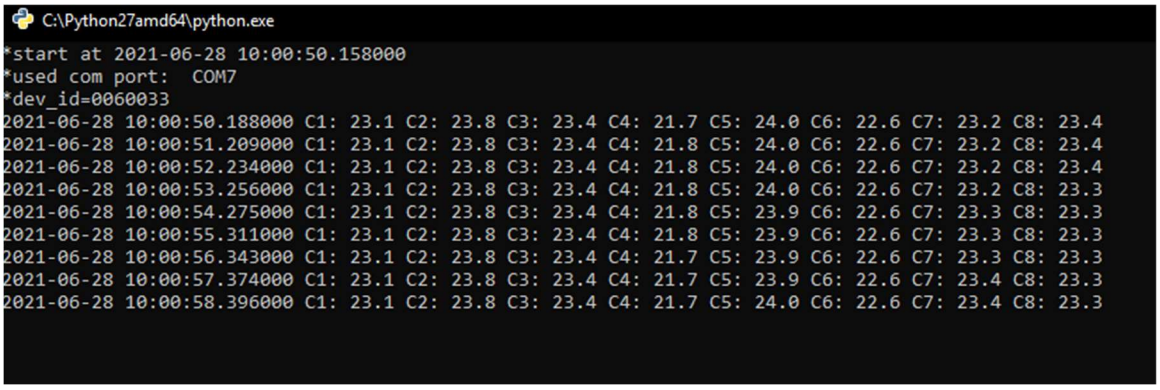

Figure 7. USB Temperature Logging

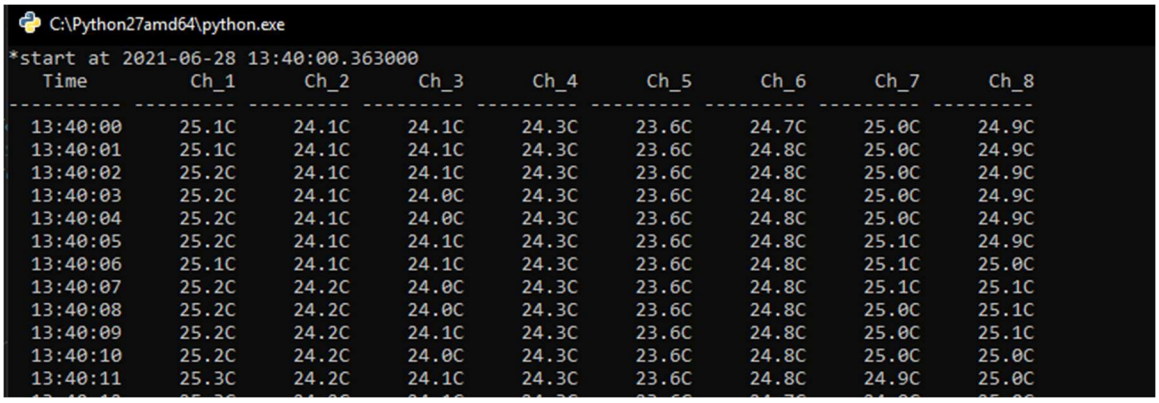

Figure 8. Modbus Temperature Logging

4.1. USB Temperature Logging

```
#!/usr/bin/env python
# coding: utf-8
# --------------------------------------------------------------------------------
# python 2.7.10 and pyserial2.7
# HW: FOTEMP-T2 USB 8CH
# --------------------------------------------------------------------------------
# SETTINGS ...
COMPORT = 'COM7'
COMPORT_TIMEOUT = 1                     # in seconds
COMPORT_SPEED = 57600                   # in baud
# --------------------------------------------------------------------------------
import serial,time
from datetime import datetime

def get_temp(ser):
    rc1=""
    ser.write("?02\r\n")
    rc1 =ser.readline()
    rc1+=ser.readline()
    rc2=rc1.split('\r')[0]
    rc3=rc2.split(' ')
    return rc3[1:]

def get_id_addr(ser):
    rc1=""
    ser.write("?41\r\n")
    rc1 =ser.readline()
    rc1+=ser.readline()
    rc2=rc1.split('\n')
    rc3=rc2[0].replace('\r','')
    rc3=rc3.replace("#41",'')
    rc3=rc3.replace(' ','')
    i=1
    rc4=""
    for char in rc3:
            if i % 2 == 0:
                    rc4 += char
            i += 1
    return rc4

if __name__ == "__main__":
    print "*start at "+datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
    ser = serial.Serial(COMPORT, COMPORT_SPEED, timeout=COMPORT_TIMEOUT)
    print "*used com port: ",ser.name
    print "*dev_id="+ get_id_addr(ser)
    while 1:
            s=datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
            temperatures=get_temp(ser)
            #print temperatures
            s=s+" C1: "+str(float(temperatures[0])/10.0)
            s=s+" C2: "+str(float(temperatures[1])/10.0)
            s=s+" C3: "+str(float(temperatures[2])/10.0)
            s=s+" C4: "+str(float(temperatures[3])/10.0)
            s=s+" C5: "+str(float(temperatures[4])/10.0)
            s=s+" C6: "+str(float(temperatures[5])/10.0)
            s=s+" C7: "+str(float(temperatures[6])/10.0)
            s=s+" C8: "+str(float(temperatures[7])/10.0)
            print s
            time.sleep(1) #sec
```

4.2. Modbus Temperature Logging

```
#!/usr/bin/env python
# coding: utf-8
# --------------------------------------------------------------------------------
# python 2.7.10 and pyserial2.7
# FOTEMP-T2 MODBUS 16CH
# --------------------------------------------------------------------------------
# SETTINGS ...
COMPORT = 'COM8'                # COM Port (for ModbusRTU)
IPAddr = 'XXX.XXX.XXX.XXX'      # IP ADDRESS (for ModbusTCP)
DevAddr = 1                     # Modbus ID
DevChCnt = 8                    # Channel Count
delay = 1                       # Measurement Delay (s)
# --------------------------------------------------------------------------------
import time,sys
from datetime import datetime
from pymodbus.constants import DeviceInformation
from pymodbus.client.sync import ModbusSerialClient as ModbusClient
#from pymodbus.client.sync import ModbusTcpClient as ModbusClient
from modbuslib import RegisterService
from modbuslib import RegTypes
from modbuslib import MB_DeviceInfo
from modbuslib import FtRegMap
from modbuslib import getflDiv10

client = ModbusClient(method='rtu', port=COMPORT, timeout=1, parity='E', baudrate=19200)
#client = ModbusClient(IPAddr.split('\r')[0], port=1312, retries=5, retry_on_empty=True)
try:
    client.connect()
    RegSrv = RegisterService(client, DevAddr)
except:
    print "No connection possible!!"
    print "Close Connection"
    client.close()
    sys.exit()

try:
  print "*start at "+datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
  print "   Time   ",
  for ch in range(1,int(DevChCnt)+1):
    print "     Ch_"+str(ch),
  print ""
  print "----------",
  for ch in range(1,int(DevChCnt)+1):
    print "---------",
  print ""
  while 1 == 1:
    zeit = time.strftime("%H:%M:%S")
    masterRspTAvg = RegSrv.execute(RegTypes.ReadInp, FtRegMap.InpTempAvg, int(DevChCnt), -1)
    print "{0:>10}".format(zeit),
    for ch in range(0,int(DevChCnt)):
      print "{0:>8}C".format(getflDiv10(masterRspTAvg.registers[ch])),
    print ""
    time.sleep(int(delay))
except StandardError, err:
  print err

finally:
  raw_input("...enter to exit")
  print "Close Connection"
  client.close()
```