Modbus Protocol-PYTHON-SDK Quick Start

Protocol Introduction

Modbus Protocol: Modbus is a serial communication protocol that was published in 1979 by Modicon Corporation (now Schneider Electric) for communication using Programmable Logic Controllers (PLCs). Modbus has become the industry standard (De facto) for communication protocols in the industrial field, and is now a common connection method between industrial electronic devices.

Routine Introduction

This routine introduces how to use Python to develop the PC software to connect to a Modbus protocol sensor, receive

sensor data and communicate with the sensor;

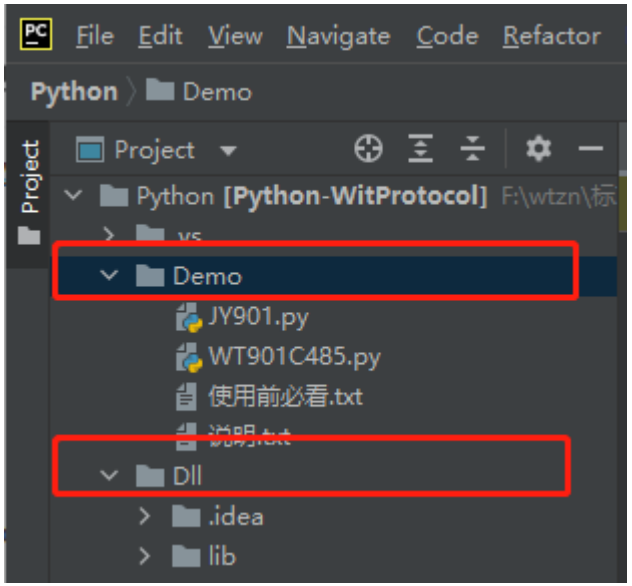
Please read the relevant sensor manual before viewing this routine to understand the protocol used by the sensor and the basic functions of the sensor

Routine Directory

The routine project directory is as follows

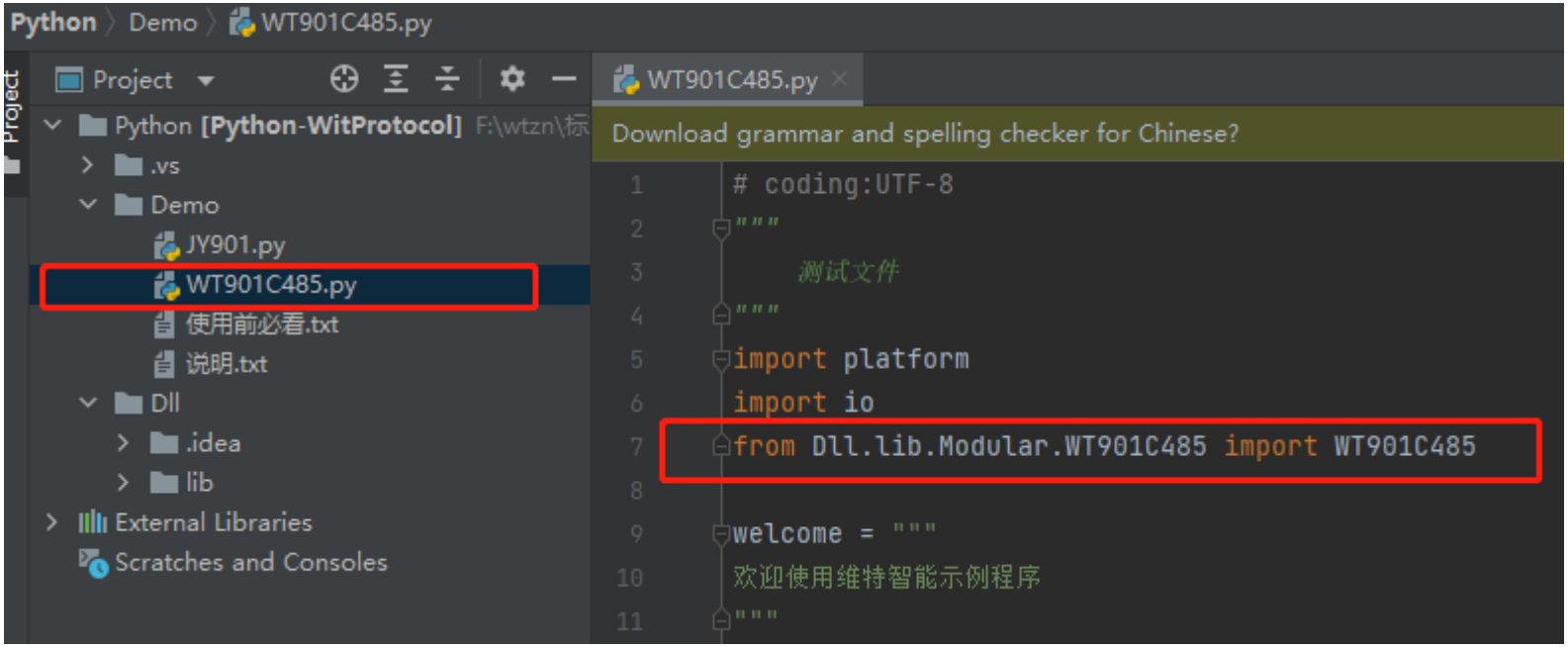
Dll: The dependency file of the project.

Demo: WT901C485.py in the routine is an example of a Modbus protocol sensor



Import dependency files

The routine project introduces dependency files as follows


Python
Demo
WT901C485.py
Project
Python [Python-WitProtocol] F:\wtzn\标
.vs
Demo
JY901.py
WT901C485.py
使用前必看.txt
说明.txt
Dll
.idea
lib
External Libraries
Scratches and Consoles
WT901C485.py
Download grammar and spelling checker for Chinese?
coding:UTF-8
"""
测试文件
"""
import platform
import io
from Dll.lib.Modular.WT901C485 import WT901C485
welcome = """
欢迎使用维特智能示例程序
"""

Initialize the device

The routine works as follows

```
if __name__ == '__main__':
    print(welcome)
    # initialize the device
    WT901C485 = WT901C485()
    if platform.system().lower() == 'linux':
        PortName = "/dev/ttyUSB0"
    else:
        PortName = "COM11"

    Baudrate = 9600
```

```
# set serial port
WT901C485.SetPortName(PortName)
# set baud rate
WT901C485.SetBaudrate(Baudrate)
# set sensor ID
WT901C485.ADDR = 0x50
# Set the read data interval (milliseconds)
WT901C485.iReadInterval = 300
# open serial port
WT901C485.Open()
if WT901C485.IsOpen():
    print("Device opened successfully")
```

```
# Add up calibration
AppliedCalibration(WT901C485)


# Magnetic field calibration
# StartFieldCalibration(WT901C485)


# read 0x03 register
# wait time
waitTime = 200
# Send read command and wait for sensor to return data
IsReadReg(WT901C485, 0X03, waitTime)
# The following line is equivalent to the above. It is recommended to use the above
# WT901C485.SendProtocolData([0x50, 0x03, 0x00, 0x03, 0x00, 0x04, 0xB9, 0x88], waitTime)
```

```
print("0x03 read result:" + str(WT901C485.GetDeviceData("0x03")))


# Write the corresponding value of the register: write register 0x03 to 0x06
WT901C485.SendWriteReg(0x03, 0x06)
# save the value of the register
WT901C485.SaveReg()
# The following line is equivalent to the above. It is recommended to use the above
# WT901C485.SendProtocolData([0x50, 0x06, 0x00, 0x03, 0x00, 0x06, 0xF4, 0x49], waitTime)
# save the value of the register
# WT901C485.SendProtocolData([0x50, 0x06, 0x00, 0x00, 0x00, 0x00, 0x84, 0x4B], waitTime)


# Bind receiving events to record data events
WT901C485.AddOnRecord(WT901C485_OnRecord)
```

```
        input()


        # close the device
        WT901C485.Close()
        # remove event
        WT901C485.RemoveOnRecord(WT901C485_OnRecord)
else:
        print("Failed to open the device")
```

Turn on the device

The WT901C485 object represents the WT901C485 device in the program, and you can communicate with the device through it; when opening the device, you need to specify the serial port number and baud rate of the sensor, and then call the WT901C485.Open() method after specifying

Turn off the device

Close the device and call the WT901C485.Close() method

Receive sensor data

Receive data

The WT901C485 object will automatically solve the sensor data and save it on itself. The sensor data can be obtained through the WT901C485.GetDeviceData() method. WT901C485.GetDeviceData() needs to pass in a key to get sensor data.

```
def WT901C485_OnRecord(deviceModel):
    """
    This is called when sensor data is refreshed and you can log data here
    :param deviceModel:
    :return:
    """
    builder = io.StringIO()
    # device name
```

```
builder.write(deviceModel.deviceName + "\n")
# chip time
builder.write("Chiptime:" + str(deviceModel.GetDeviceData("Chiptime"))+"\t")
# temperature
builder.write("Temperature:" + str(deviceModel.GetDeviceData("Temperature"))+"\n")
# acceleration
builder.write("AccX:" + str(deviceModel.GetDeviceData("AccX"))+"g \t")
builder.write("AccY:" + str(deviceModel.GetDeviceData("AccY"))+"g \t")
builder.write("AccZ:" + str(deviceModel.GetDeviceData("AccZ"))+"g \n")
# angular velocity
builder.write("GyroX:" + str(deviceModel.GetDeviceData("GyroX"))+"°/s \t")
builder.write("GyroY:" + str(deviceModel.GetDeviceData("GyroY"))+"°/s \t")
builder.write("GyroZ:" + str(deviceModel.GetDeviceData("GyroZ"))+"°/s \n")
```

```
# angle
builder.write("AngleX:" + str(deviceModel.GetDeviceData("AngleX"))+"° \t")
builder.write("AngleY:" + str(deviceModel.GetDeviceData("AngleY"))+"° \t")
builder.write("AngleZ:" + str(deviceModel.GetDeviceData("AngleZ"))+"° \n")
# Magnetic field
builder.write("MagX:" + str(deviceModel.GetDeviceData("MagX"))+"uT \t")
builder.write("MagY:" + str(deviceModel.GetDeviceData("MagY"))+"uT \t")
builder.write("MagZ:" + str(deviceModel.GetDeviceData("MagZ"))+"uT \n")
# latitude and longitude
builder.write("Lon:" + str(deviceModel.GetDeviceData("Lon"))+"\t")
builder.write("Lat:" + str(deviceModel.GetDeviceData("Lat"))+"\n")
# barometric pressure and altitude
builder.write("Pressure:" + str(deviceModel.GetDeviceData("Pressure"))+"\t")
```

```
builder.write("Height:" + str(deviceModel.GetDeviceData("Height"))+"\n")
# GPS: altitude, heading angle, GPS ground speed
builder.write("GPSHeight:" + str(deviceModel.GetDeviceData("GPSHeight")) + "\t")
builder.write("GPSYaw:" + str(deviceModel.GetDeviceData("GPSYaw"))+"\t")
builder.write("GPSV:" + str(deviceModel.GetDeviceData("GPSV")) + "\n")
# Four elements
builder.write("Q0:" + str(deviceModel.GetDeviceData("Q0"))+"\t")
builder.write("Q1:" + str(deviceModel.GetDeviceData("Q1"))+"\t")
builder.write("Q2:" + str(deviceModel.GetDeviceData("Q2"))+"\t")
builder.write("Q3:" + str(deviceModel.GetDeviceData("Q3"))+"\n")
# The port number
builder.write("D0:" + str(deviceModel.GetDeviceData("D0"))+"\t")
builder.write("D1:" + str(deviceModel.GetDeviceData("D1"))+"\t")
```

```
builder.write("D2:" + str(deviceModel.GetDeviceData("D2"))+"\t")

builder.write("D3:" + str(deviceModel.GetDeviceData("D3"))+"\n")

# Positioning accuracy: number of satellites, position accuracy, horizontal accuracy, vertical accuracy

builder.write("SVNUM:" + str(deviceModel.GetDeviceData("SVNUM")) + "\t")

builder.write("PDOP:" + str(deviceModel.GetDeviceData("PDOP"))+"\t")

builder.write("HDOP:" + str(deviceModel.GetDeviceData("HDOP")) + "\t")

builder.write("VDOP:" + str(deviceModel.GetDeviceData("VDOP"))+"\n")

# version number

builder.write("VersionNumber:" + str(deviceModel.GetDeviceData("VersionNumber")) + "\n")


print(builder.getvalue())
```

Record data

The data of the sensor can be obtained through the WT901C485 object, but usually the host computer needs to record the data of the sensor. The WT901C485 has an OnRecord event that will inform you when to record the data, and the OnRecord event can be realized when the device is turned on;) method to record data

Config the sensor

The sensor can be operated by the method of WT901C485

WT901C485.UnlockReg() Send unlock register command

WT901C485.AppliedCalibration() Sends the addition calibration command

WT901C485.StartFieldCalibration() Send start field calibration command

WT901C485.EndFieldCalibration() Send end field calibration command

WT901C485.SendProtocolData() Send other commands

Acceleration calibration

Add-on calibration of the sensor by calling the WT901C485.AppliedCalibration() method

```
def AppliedCalibration(WT901C485):
    """
    Addition calibration
    :param WT901C485: Device model
    :return:
    """
    if WT901C485.IsOpen():
        # Unlock the register and send the command
        WT901C485.UnlockReg()
```

```
        # Add up calibration
        WT901C485.AppliedCalibration()

        # The following two lines are equivalent to the above, it is recommended to use the above
        # Unlock the register and send the command
        # WT901C485.SendProtocolData([0x50, 0x06, 0x00, 0x69, 0xB5, 0x88,0x22,0xA1], 50)
        # Add up calibration
        # WT901C485.SendProtocolData([50, 0x06, 0x00, 0x01, 0x00, 0x01, 0x32, 0x4B], 4000)

        print("Completion of total calibration")
    else:
        print("The device is not open")
```

Magnetic Field Calibration

Magnetic field calibration of the sensor by calling the WT901C485.StartFieldCalibration() method and the WT901C485.EndFieldCalibration() method

```
def StartFieldCalibration(WT901C485):
    """
    Magnetic Field Calibration
    :param WT901C485: Device model
    :return:
    """
    if WT901C485.IsOpen():
        # Unlock the register and send the command
        WT901C485.UnlockReg()
        # start magnetic field calibration
```

```
        WT901C485.StartFieldCalibration()

        # The following two lines are equivalent to the above, it is recommended to use the above
        # Unlock the register and send the command
        # WT901C485.SendProtocolData([0x50, 0x06, 0x00, 0x69, 0xB5, 0x88,0x22 , 0xA1)
        # start magnetic field calibration
        # WT901C485.SendProtocolData([0x50, 0x06, 0x00, 0x01, 0x00, 0x07, 0x94, 0x49], 100)

        if input("Please make a slow rotation around the XYZ axis respectively, after the three-axis rotation is completed,
end the calibration (Y/N)?").lower() == "y":
            # Unlock the register and send the command
            WT901C485.UnlockReg()
            # end magnetic field calibration
```

```
            WT901C485.EndFieldCalibration()


            # The following two lines are equivalent to the above, it is recommended to use the above
            # Unlock the register and send the command
            # WT901C485.SendProtocolData([0x50, 0x06, 0x00, 0x69, 0xB5, 0x88,0x22,0xA1], 50)
            # start magnetic field calibration
            # WT901C485.SendProtocolData([0x50, 0x06, 0x00, 0x01, 0x00, 0x00, 0xD5, 0x8B], 100)

            print("End magnetic field calibration")
else:
        print("The device is not open")
```

More

For other operations, please refer to the sensor manual

Read sensor register

The sensor register can be read through the WT901C485.SendReadReg() method, or the WT901C485.SendProtocolData() method can be used

After sending the read command, the register value will be saved in the WT901C485, you need to get the register data through WT901C485.GetDeviceData()

```
def IsReadReg(WT901C485, reg, waitTime):
    """
    read register
```

```
:param WT901C485: Device model
:param reg: register address
:param waitTime: wait time
:return:
"""
bRet = False
if WT901C485.IsOpen():
    # read register
    if WT901C485.SendReadReg(reg, waitTime):
        bRet = True
    else:
        print(str(reg) + "Failed to read")
else:
```

```
        print("The device is not open")


    return bRet
```

WT901C485 API

Method	Directions	Parameter introduction	Return value
void SetPortName(string portName)	Set the serial port to be opened	portName: serial port number	void

void SetBaudrate(int baudRate)	Specifies the baud rate to be turned on	baudRate: baud rate	void
void Open() Open	Turn on the device	NO	void
bool IsOpen()	Whether the device is open	NO	Return whether to open open: true off: false
void Close()	Turn off the device	NO	void
void SendProtocolData(byte[] data, int waitTime)	Send the data with the	data: data to be sent	void

	protocol and specify the waiting time	waitTime: wait time	
void SendReadReg(byte reg, int waitTime)	Send the command to read the register	reg: command to be sent wait time: wait time	void
void UnlockReg()	unlock register	NO	void
void SaveReg()	save register	N0	void

void AppliedCalibration()	Acceleration calibration	NO	void
void StartFieldCalibration()	Start magnetic field calibration	NO	void
void EndFieldCalibration()	end magnetic field	NO	void
Void SetModbusId(byte modbusId)	Specifies the address	modbusId: Modbus address	void
string GetDeviceName()	get device name	NO	return

			device name
string GetDeviceData(string key)	Get key value data	key: data key value	return data value