

High Precision Modbus Protocol SDK Quick Guide

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.

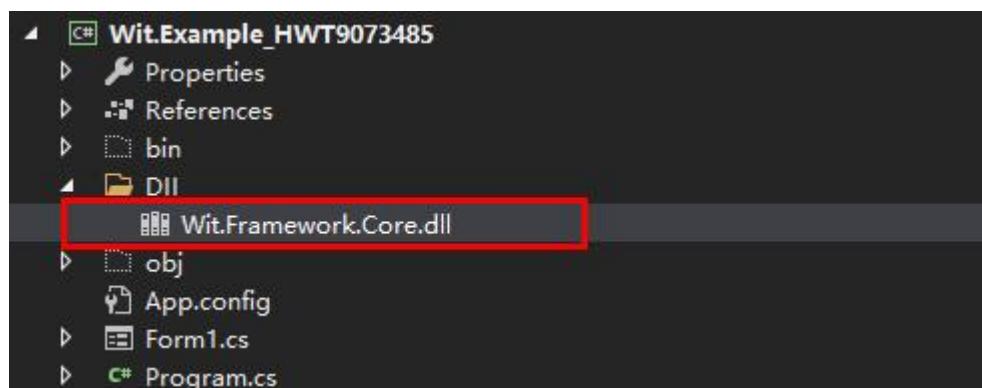
This routine introduces how to use C# to develop the host computer to connect the High Precision Modbus protocol sensor, receive sensor data, and communicate with the sensor; Please read the relevant sensor manual before viewing this routine to understand the protocol used by the sensor and the basic functions of the sensor.

Routine directory

The routine project directory is as follows

Dll: The dependency files of the project, please import these dlls into your project before running the project.

Form1: There is only one Form window in the routine, all logic codes are in the Form window file, and there are no other files



Turn on the device

The HWT9073485 object represents the HWT9073485 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 call the HWT9073485.Open() method after setting it.

```
/// <summary>
/// Turn on the device
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void openButton_Click(object sender, EventArgs e)
{
    // Get the serial number and baud rate when connect
    string portName;
    int baudrate;
    byte modbusId;
    try
    {
        portName = (string)portComboBox.SelectedItem;
        baudrate = (int)baudComboBox.SelectedItem;
        modbusId = byte.Parse(ModbusTextBox.Text.Replace("0x", ""),
            System.Globalization.NumberStyles.HexNumber);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

    // Do not open again
    if (HWT9073485.isOpen())
    {
        return;
    }

    // Turn on the device
    try
    {
        HWT9073485.SetPortName(portName);
        HWT9073485.SetBaudrate(baudrate);
        HWT9073485.SetModbusId(modbusId);
    }
```

```

        HWT9073485.Open();
        // Implement logging data events
        HWT9073485.OnRecord += HWT9073485_OnRecord;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

```

Turn off the device

Close the device and call the HWT9073485.Close() method.

```

///<summary>
/// Turn off the device
///</summary>
///<param name="sender"></param>
///<param name="e"></param>
private void closeButton_Click(object sender, EventArgs e)
{
    try
    {
        // Turn off the device if it is already on
        if (HWT9073485.IsOpen())
        {
            HWT9073485.OnRecord -= HWT9073485_OnRecord;
            HWT9073485.Close();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

```

Receive sensor data

GET DATA

The HWT9073485 object will automatically solve the sensor data and save it on itself.

The sensor data can be obtained through the HWT9073485.GetDeviceData() method.

HWT9073485.GetDeviceData() needs to pass in a key to get sensor data. Please check the key used in the routine, the keys are all stored in the WitSensorKey class.

```
/// <summary>
/// Get device data
/// </summary>
private string GetDeviceData(HWT9073485 HWT9073485)
{
    StringBuilder builder = new StringBuilder();
    builder.Append(HWT9073485.GetDeviceName()).Append("\n");
    // acceleration

    builder.Append("AccX").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKe
y.AccX)).Append("g \t");

    builder.Append("AccY").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKe
y.AccY)).Append("g \t");

    builder.Append("AccZ").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKe
y.AccZ)).Append("g \n");
    // Angular velocity

    builder.Append("GyroX").Append(":").Append(HWT9073485.GetDeviceData(WitSensorK
ey.AsX)).Append("°/s \t");

    builder.Append("GyroY").Append(":").Append(HWT9073485.GetDeviceData(WitSensorK
ey.AsY)).Append("°/s \t");

    builder.Append("GyroZ").Append(":").Append(HWT9073485.GetDeviceData(WitSensorK
ey.AsZ)).Append("°/s \n");
    // angle

    builder.Append("AngleX").Append(":").Append(HWT9073485.GetDeviceData(WitSensor
Key.AngleX)).Append("° \t");

    builder.Append("AngleY").Append(":").Append(HWT9073485.GetDeviceData(WitSensor
Key.AngleY)).Append("° \t");
```

```
builder.Append("AngleZ").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.AngleZ)).Append("° \n");
// magnetic field

builder.Append("MagX").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.HX)).Append("uT \t");

builder.Append("MagY").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.HY)).Append("uT \t");

builder.Append("MagZ").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.HZ)).Append("uT \n");
// latitude and longitude

builder.Append("Lon").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Lon)).Append("° \t");

builder.Append("Lat").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Lat)).Append("° \n");
// port number

builder.Append("D0").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.D0)).Append("\t");

builder.Append("D1").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.D1)).Append("\t");

builder.Append("D2").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.D2)).Append("\t");

builder.Append("D3").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.D3)).Append("\n");
// Quaternion

builder.Append("Q0").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Q0)).Append("\t");

builder.Append("Q1").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Q1)).Append("\t");

builder.Append("Q2").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Q2)).Append("\t");
```

```
builder.Append("Q3").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Q3)).Append("\n");
// air pressure

builder.Append("P").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Q1)).Append("Pa \t");

builder.Append("H").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.Q2)).Append("m \t");
// temperature

builder.Append("T").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.T));
Append("°C \n");
// GPS

builder.Append("GPSHeight").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.GPSHeight)).Append(" m \t");

builder.Append("GPSTime").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.GPSTime));
Append("° \t");

builder.Append("GPSV").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.GPSV)).Append("km/h \n");
// positioning accuracy

builder.Append("PDOP").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.PDOP)).Append("\t");

builder.Append("VDOP").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.VDOP)).Append("\t");

builder.Append("HDOP").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.HDOP)).Append("\n");
// version number

builder.Append("VersionNumber").Append(":").Append(HWT9073485.GetDeviceData(WitSensorKey.VersionNumber)).Append("\n");
return builder.ToString();
}
```

RECORD DATA

The data of the sensor can be obtained through the HWT9073485 object, but usually the host computer needs to record the data of the sensor. The HWT9073485 has an OnRecord event to notify you when to record the data, and the OnRecord event can be implemented when the device is turned on; Then record data by cooperating with HWT9073485.GetDeviceData() method.

```
/// <summary>
/// Turn on the device
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void openButton_Click(object sender, EventArgs e)
{
    // Get the serial number and baud rate when connect
    string portName;
    int baudrate;
    byte modbusId;
    try
    {
        portName = (string)portComboBox.SelectedItem;
        baudrate = (int)baudComboBox.SelectedItem;
        modbusId = byte.Parse(ModbustextBox.Text.Replace("0x", ""),
            System.Globalization.NumberStyles.HexNumber);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

    // Do not open again
    if (HWT9073485.IsOpen())
    {
        return;
    }

    // Turn on the device
    try
    {
        HWT9073485.SetPortName(portName);
```

```

        HWT9073485.SetBaudrate(baudrate);
        HWT9073485.SetModbusId(modbusId);
        HWT9073485.Open();
        // Implement logging data events
        HWT9073485.OnRecord += HWT9073485_OnRecord;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

/// <summary>
/// This is called when sensor data is refreshed and you can log data here
/// </summary>
/// <param name="HWT9073485"></param>
private void HWT9073485_OnRecord(HWT9073485 HWT9073485)
{
    string text = GetDeviceData(HWT9073485);
    Debug.WriteLine(text);
}

```

Set up the sensor

The sensor can be operated by the method of HWT9073485

- HWT9073485.UnlockReg() Send unlock register command
- HWT9073485.AppliedCalibration() Sends the addition calibration command
- HWT9073485.StartFieldCalibration() Send start field calibration command
- HWT9073485.EndFieldCalibration() Send end field calibration command
- HWT9073485.EndFieldCalibration() Send end field calibration command
- HWT9073485.SetBandWidth() Send command to set bandwidth
- HWT9073485.SendProtocolData() Send other commands

SET ADDRESS

Set the sensor address by calling the HWT9073485.SetModbusId() method

```

/// <summary>
/// Set the device address to 50
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SetAddrBtn_Click(object sender, EventArgs e)
{
    if (HWT9073485.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        HWT9073485.UnlockReg();
        byte modbusId = byte.Parse(Modbus(textBox.Text.Replace("0x", ""), System.Globalization.NumberStyles.HexNumber));
        HWT9073485.SetModbusId(modbusId);
        // The following two lines are equivalent to the above, and it is recommended to
        // use the above

        //HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
        busId(), 0x69, 0xB588));
        //HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

        //HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
        busId(), 0x1A, 0x0050));
        //HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 1A, 00, 50, A1, B0 });

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

ACCELERATION CALIBRATION

Acceleration calibration of the sensor by calling the HWT9073485.AppliedCalibration() method

```

/// <summary>
/// Acceleration calibration
/// </summary>

```

```

/// <param name="sender"></param>
/// <param name="e"></param>
private void appliedCalibrationButton_Click(object sender, EventArgs e)
{
    if (HWT9073485.IsOpen() == false)
    {
        return;
    }

    try
    {
        // Unlock registers and send commands
        HWT9073485.UnlockReg();
        HWT9073485.AppliedCalibration();
        // The following two lines are equivalent to the above, and it is recommended to
        // use the above

        //HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
        busId(), 0x69, 0xB588));
        //HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

        //HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
        busId(), 0x01, 0x0001));
        //HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 01, 00, 01, 32, 4B });
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

MAGNETIC FIELD CALIBRATION

Perform magnetic field calibration on the sensor by calling the HWT9073485.StartFieldCalibration() method and the HWT9073485.EndFieldCalibration() method

```

/// <summary>
/// Start magnetic field calibration
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void startFieldCalibrationButton_Click(object sender, EventArgs e)
{
    if (HWT9073485.IsOpen() == false)

```

```

{
    return;
}
try
{
    // Unlock registers and send commands
    HWT9073485.UnlockReg();
    HWT9073485.StartFieldCalibration();
    // The following two lines are equivalent to the above, and it is recommended to
    // use the above

    //HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
    busId(), 0x69, 0xB588));
    //HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

    //HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
    busId(), 0x01, 0x0007));
    //HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 01, 00, 07, 94, 49 });
    MessageBox.Show("To start the magnetic field calibration, please make one turn
        around each of the XYZ axes of the sensor, and click [End Magnetic Field
        Calibration] after completing the rotation.");
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

/// <summary>
/// end magnetic field calibration
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void endFieldCalibrationButton_Click(object sender, EventArgs e)
{
    if (HWT9073485.isOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        HWT9073485.UnlockReg();
        HWT9073485.EndFieldCalibration();
    }
}

```

```

// The following two lines are equivalent to the above, and it is recommended to
use the above

//HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
busId(), 0x69, 0xB588));
//HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

//HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetMod
busId(), 0x01, 0x0000));
//HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 01, 00, 00, D5, 8B });
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

SET BANDWIDTH

Set the bandwidth of the sensor by calling HWT9073485. SetBandWidth() method.

```

/// <summary>
/// Set the bandwidth 20Hz
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void bandWidth20_Click(object sender, EventArgs e)
{
    if (HWT9073485.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        HWT9073485.UnlockReg();
        HWT9073485.SetBandWidth(0x04);
        // The following two lines are equivalent to the above, and it is recommended to
use the above

//HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetModbusId(),
0x69, 0xB588));
//HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

```

```

//HWT9073485.SendProtocolData(Modbus16Utils.GetWrite(HWT9073485.GetModbusId(),
0x01, 0x0000));
    //HWT9073485.SendProtocolData(new byte[] { 50, 06, 00, 1F, 00, 04, D5, 8B });
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

Read sensor register

The sensor register can be read through the HWT9073485.SendReadReg() method, or the HWT9073485.SendProtocolData() method can be used

After sending the read command, the register value will be saved in HWT9073485, and the register data needs to be obtained through HWT9073485.GetDeviceData().

```

/// <summary>
/// read 03 register
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void readReg03Button_Click(object sender, EventArgs e)
{
    if (HWT9073485.IsOpen() == false)
    {
        return;
    }
    try
    {
        // wait time
        int waitTime = 150;
        // Send a read command and wait for the sensor to return data. If you don't read it,
        // you can extend the waitTime or read it several times.
        HWT9073485.SendReadReg(0x03, waitTime);
        // The following two lines are equivalent to the above, and it is recommended to
        // use the above
    }
}

```

```
//HWT9073485.SendProtocolData(Modbus16Utils.GetRead(HWT9073485.GetMod
busId(), 0x03, 0x01), waitTime);
//HWT9073485.SendProtocolData(new byte[] { 50, 03, 00, 03, 00, 01, 79, 8B },
waitTime);

string reg03Value = HWT9073485.GetDeviceData("03");
MessageBox.Show($"The value of register 03 is : {reg03Value}");
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

}
```

Method	Directions	Parameter introduction	Return value
void SetPortName(string portName)	set serial port	portName: serial number	void
void SetBaudrate(int baudRate)	Specify the baud rate to turn on	baudRate: Baud rate	void
void Open()	Turn on the device	NO	void
bool IsOpen()	Is the device turned on	NO	Return whether to open open: true off: false
void Close()	Turn off the device	NO	void
void SendData(byte[] data, out byte[] returnData, bool isWaitReturn, int waitTime , int repetition)	send data	data: data to be sent returnData: the data returned by the sensor isWaitReturn: Whether the sensor needs to return data waitTime: time to wait for the sensor to return data, in ms, default 100ms repetition: repeated sending times	void
void SendProtocolData(byte[] data)	Send data with protocol	data: data to be sent	void
void SendProtocolData(byte[] data, int waitTime)	Send the data with the protocol and specify the waiting time	data: data to be sent waitTime: wait time	void
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	NO	void
void AppliedCalibration()	Acceleration calibration	NO	void

void StartFieldCalibration()	Start magnetic field calibration	NO	void
void EndFieldCalibration()	end magnetic field calibration	NO	void
void SetBandWidth(byte band)	Set bandwidth	Band: the bandwidth to be set	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
void SetModbusId(byte modbusId)	Specify address	modbusId: Modbus address	void
byte GetModbusId()	Get device address	NO	return device address