

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 Modbus protocol sensor, receive sensor data, and communicate with the sensor;

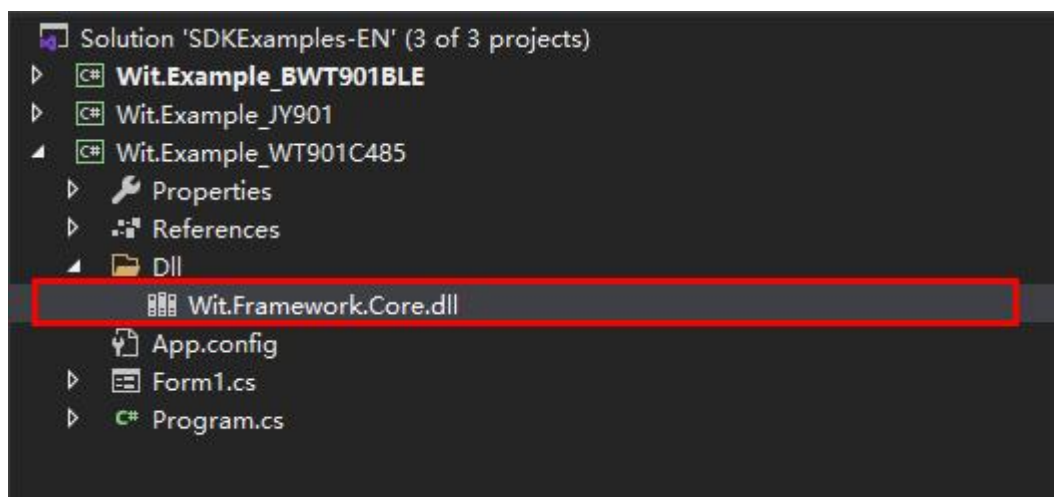
Please read the relevant sensor manual before viewing this routine to understand the protocol used by the sensor and the basic functions of the sensor.

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 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 call the WT901C485.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 (WT901C485.IsOpen())
    {
        return;
    }
}
```

```

// Turn on the device
try
{
    WT901C485.SetPortName(portName);
    WT901C485.SetBaudrate(baudrate);
    WT901C485.SetModbusId(modbusId);
    WT901C485.Open();
    // Implement logging data events
    WT901C485.OnRecord += WT901C485_OnRecord;
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
    return;
}
}

```

Turn off the device

Close the device and call the WT901C485.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 (WT901C485.IsOpen())
        {
            WT901C485.OnRecord -= WT901C485_OnRecord;
            WT901C485.Close();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

```

```
}  
}
```

Receive sensor data

GET DATA

The WT901C485object 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. 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(WT901C485 WT901C485)  
{  
    StringBuilder builder = new StringBuilder();  
    builder.Append(WT901C485.GetDeviceName()).Append("\n");  
    // acceleration  
  
    builder.Append("AccX").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.  
    AccX)).Append("g \t");  
  
    builder.Append("AccY").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.  
    AccY)).Append("g \t");  
  
    builder.Append("AccZ").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.  
    AccZ)).Append("g \n");  
    // Angular velocity  
  
    builder.Append("GyroX").Append(":").Append(WT901C485.GetDeviceData(WitSensorKe  
    y.AsX)).Append("°/s \t");  
  
    builder.Append("GyroY").Append(":").Append(WT901C485.GetDeviceData(WitSensorKe  
    y.AsY)).Append("°/s \t");  
  
    builder.Append("GyroZ").Append(":").Append(WT901C485.GetDeviceData(WitSensorKe  
    y.AsZ)).Append("°/s \n");  
}
```

// angle

```
builder.Append("AngleX").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.AngleX)).Append("° \t");
```

```
builder.Append("AngleY").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.AngleY)).Append("° \t");
```

```
builder.Append("AngleZ").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.AngleZ)).Append("° \n");
```

// magnetic field

```
builder.Append("MagX").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.HX)).Append("uT \t");
```

```
builder.Append("MagY").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.HY)).Append("uT \t");
```

```
builder.Append("MagZ").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.HZ)).Append("uT \n");
```

// latitude and longitude

```
builder.Append("Lon").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Lon)).Append("° \t");
```

```
builder.Append("Lat").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Lat)).Append("° \n");
```

// port number

```
builder.Append("D0").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.D0)).Append("\t");
```

```
builder.Append("D1").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.D1)).Append("\t");
```

```
builder.Append("D2").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.D2)).Append("\t");
```

```
builder.Append("D3").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.D3)).Append("\n");
```

// Quaternion

```
builder.Append("Q0").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Q0)).Append("\t");
```

```
builder.Append("Q1").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Q1)).Append("\t");
```

```
builder.Append("Q2").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Q2)).Append("\t");
```

```
builder.Append("Q3").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Q3)).Append("\n");  
// air pressure
```

```
builder.Append("P").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Q1)).Append("Pa \t");
```

```
builder.Append("H").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.Q2)).Append("m \t");  
// temperature
```

```
builder.Append("T").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.T)).Append("°C \n");  
// GPS
```

```
builder.Append("GPSHeight").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.GPSHeight)).Append(" m \t");
```

```
builder.Append("GPSYaw").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.GPSYaw)).Append("° \t");
```

```
builder.Append("GPSV").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.GPSV)).Append("km/h \n");  
// positioning accuracy
```

```
builder.Append("PDOP").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.PDOP)).Append("\t");
```

```
builder.Append("VDOP").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.VDOP)).Append("\t");
```

```
builder.Append("HDOP").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.HDOP)).Append("\n");  
// version number
```

```
builder.Append("VersionNumber").Append(":").Append(WT901C485.GetDeviceData(WitSensorKey.VersionNumber)).Append("\n");
```

```

        return builder.ToString();
    }

```

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 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 WT901C485.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 (WT901C485.IsOpen())

```

```

    {
        return;
    }

    // Turn on the device
    try
    {
        WT901C485.SetPortName(portName);
        WT901C485.SetBaudrate(baudrate);
        WT901C485.SetModbusId(modbusId);
        WT901C485.Open();
        // Implement logging data events
        WT901C485.OnRecord += WT901C485_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="WT901C485"></param>
private void WT901C485_OnRecord(WT901C485 WT901C485)
{
    string text = GetDeviceData(WT901C485);
    Debug.WriteLine(text);
}

```

Set up 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.SetBandWidth() Send command to set bandwidth
- WT901C485.SendProtocolData() Send other commands

SET ADDRESS

Set the sensor address by calling the WT901C485.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 (WT901C485.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        WT901C485.UnlockReg();
        byte modbusId = byte.Parse(ModbusTextBox.Text.Replace("0x", ""),
            System.Globalization.NumberStyles.HexNumber);
        WT901C485.SetModbusId(modbusId);
        // The following two lines are equivalent to the above, and it is recommended to
        use the above

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        ()), 0x69, 0xB588));
        //WT901C485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        ()), 0x1A, 0x0050));
        //WT901C485.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 WT901C485.AppliedCalibration() method

```

/// <summary>
/// Acceleration calibration
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void appliedCalibrationButton_Click(object sender, EventArgs e)
{
    if (WT901C485.IsOpen() == false)
    {
        return;
    }

    try
    {
        // Unlock registers and send commands
        WT901C485.UnlockReg();
        WT901C485.AppliedCalibration();
        // The following two lines are equivalent to the above, and it is recommended to
        use the above

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        (), 0x69, 0xB588));
        //WT901C485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        (), 0x01, 0x0001));
        //WT901C485.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 WT901C485.
StartFieldCalibration() method and the WT901C485.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 (WT901C485.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        WT901C485.UnlockReg();
        WT901C485.StartFieldCalibration();
        // The following two lines are equivalent to the above, and it is recommended to
        use the above

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        ( ), 0x69, 0xB588));
        //WT901C485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        ( ), 0x01, 0x0007));
        //WT901C485.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 (WT901C485.IsOpen() == false)
    {
        return;
    }
}

```

```

try
{
    // Unlock registers and send commands
    WT901C485.UnlockReg();
    WT901C485.EndFieldCalibration();
    // The following two lines are equivalent to the above, and it is recommended to
    use the above

    //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
    ( ), 0x69, 0xB588));
    //WT901C485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

    //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
    ( ), 0x01, 0x0000));
    //WT901C485.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 WT901C485. 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 (WT901C485.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        WT901C485.UnlockReg();
        WT901C485.SetBandWidth(0x04);
        // The following two lines are equivalent to the above, and it is recommended to
        use the above
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

```

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        (), 0x69, 0xB588));
        //WT901C485.SendProtocolData(new byte[] { 50, 06, 00, 69, B5, 88, 22, A1 });

        //WT901C485.SendProtocolData(Modbus16Utils.GetWrite(T901C485.GetModbusId
        (), 0x01, 0x0000));
        //WT901C485.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 WT901C485.SendReadReg() method, or the WT901C485.SendProtocolData() method can be used

After sending the read command, the register value will be saved in WT901C485, and the register data needs to be obtained through WT901C485.GetDeviceData().

```

/// <summary>
/// read 03 register
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void readReg03Button_Click(object sender, EventArgs e)
{
    if (WT901C485.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.
    WT901C485.SendReadReg(0x03, waitTime);
    // The following two lines are equivalent to the above, and it is recommended to
    use the above

    //WT901C485.SendProtocolData(Modbus16Utils.GetRead(WT901C485.GetModbus
    Id(), 0x03, 0x01), waitTime);
    //WT901C485.SendProtocolData(new byte[] { 50, 03, 00, 03, 00, 01, 79, 8B },
    waitTime);

    string reg03Value = WT901C485.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