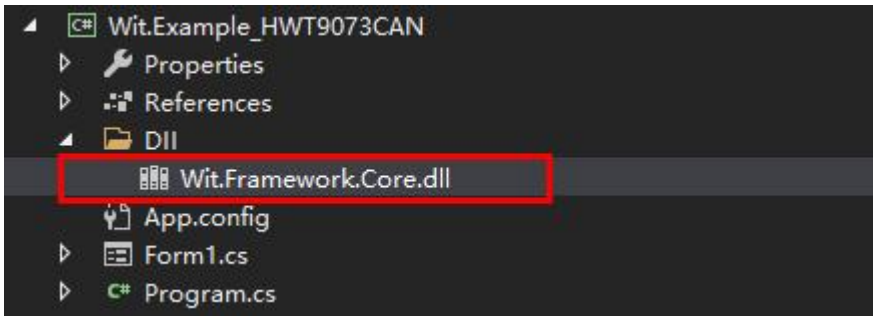
High Precision CAN Protocol SDK Quick Guide

CAN Protocol: CAN is an ISO international standardized serial communication protocol. In North America and Western Europe, the CAN bus protocol has become the standard bus for automotive computer control systems and embedded industrial control area networks, and has the J1939 protocol designed for large trucks and heavy industrial machinery vehicles with CAN as the underlying protocol.

This routine introduces how to use C# to develop the host computer to connect the CAN 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 HWT9073CAN object represents the HWT9073CAN 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 HWT9073CAN.Open() method after specifying 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 CANId;
    try
    {
        portName = (string)portComboBox.SelectedItem;
        baudrate = (int)baudComboBox.SelectedItem;
        CANId = byte.Parse(ModbustextBox.Text.Replace("0x", ""),
        System.Globalization.NumberStyles.HexNumber);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

    // Do not open again
    if (HWT9073CAN.IsOpen())
    {
        return;
    }

    // Turn on the device
    try
    {
        HWT9073CAN.SetPortName(portName);
        HWT9073CAN.SetBaudrate(baudrate);
        HWT9073CAN.SetCANId(CANId);
```

```
            HWT9073CAN.Open();
            // Implement logging data events
            HWT9073CAN.OnRecord += HWT9073CAN_OnRecord;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
}
```

Turn off the device

Close the device and call the HWT9073CAN.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 (HWT9073CAN.IsOpen())
            {
                HWT9073CAN.OnRecord -= HWT9073CAN_OnRecord;
                HWT9073CAN.Close();
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
}
```

Receive sensor data

GET DATA

The HWT9073CAN object will automatically solve the sensor data and save it on itself. The sensor data can be obtained through the HWT9073CAN.GetDeviceData() method. HWT9073CAN.GetDeviceData() needs to pass in a key to get sensor data. Please check the key to be used in the routine, the keys are all stored in the WitSensorKey class

```
/// <summary>
/// Get device data
/// </summary>
private string GetDeviceData(HWT9073CAN HWT9073CAN)
{
    StringBuilder builder = new StringBuilder();
    builder.Append(HWT9073CAN.GetDeviceName()).Append("\n");
    // acceleration

    builder.Append("AccX").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorK
    ey.AccX)).Append("g \t");

    builder.Append("AccY").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorK
    ey.AccY)).Append("g \t");

    builder.Append("AccZ").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorK
    ey.AccZ)).Append("g \n");
    // Angular velocity

    builder.Append("GyroX").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensor
    Key.AsX)).Append("°/s \t");

    builder.Append("GyroY").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensor
    Key.AsY)).Append("°/s \t");

    builder.Append("GyroZ").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensor
    Key.AsZ)).Append("°/s \n");
    // angle

    builder.Append("AngleX").Append(":").Append(HWT9073CAN.GetDeviceData(WitSenso
    rKey.AngleX)).Append("°  \t");
```

```
builder.Append("AngleY").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.AngleY)).Append("°  \t");

builder.Append("AngleZ").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.AngleZ)).Append("°  \n");
// magnetic field

builder.Append("MagX").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.HX)).Append("uT \t");

builder.Append("MagY").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.HY)).Append("uT \t");

builder.Append("MagZ").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.HZ)).Append("uT \n");
// latitude and longitude

builder.Append("Lon").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.Lon)).Append("'  \t");

builder.Append("Lat").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.Lat)).Append("'  \n");
// port number

builder.Append("D0").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.D0)).Append("\t");

builder.Append("D1").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.D1)).Append("\t");

builder.Append("D2").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.D2)).Append("\t");

builder.Append("D3").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.D3)).Append("\n");
// Quaternion

builder.Append("Q0").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.Q0)).Append("\t");

builder.Append("Q1").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.Q1)).Append("\t");
```

```
        builder.Append("Q2").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.
        Q2)).Append("\t");

        builder.Append("Q3").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.
        Q3)).Append("\n");
        // air pressure

        builder.Append("P").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.
        Q1)).Append("Pa \t");

        builder.Append("H").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.
        Q2)).Append("m \t");
        // temperature

        builder.Append("T").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorKey.T)
        ).Append("℃  \n");
        // GPS

        builder.Append("GPSHeight").Append(":").Append(HWT9073CAN.GetDeviceData(WitSe
        nsorKey.GPSHeight)).Append(" m \t");

        builder.Append("GPSYaw").Append(":").Append(HWT9073CAN.GetDeviceData(WitSens
        orKey.GPSYaw)).Append("°  \t");

        builder.Append("GPSV").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensorK
        ey.GPSV)).Append("km/h \n");
        // positioning accuracy

        builder.Append("PDOP").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensor
        Key.PDOP)).Append("\t");

        builder.Append("VDOP").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensor
        Key.VDOP)).Append("\t");

        builder.Append("HDOP").Append(":").Append(HWT9073CAN.GetDeviceData(WitSensor
        Key.HDOP)).Append("\n");
        // version number

        builder.Append("VersionNumber").Append(":").Append(HWT9073CAN.GetDeviceData(
        WitSensorKey.VersionNumber)).Append("\n");
        return builder.ToString();
    }
```

RECORD DATA

The data of the sensor can be obtained through the HWT9073CAN object, but usually the host computer needs to record the data of the sensor. HWT9073CAN has an OnRecord event that will notify you when the data should be recorded, and the OnRecord event can be realized when the device is turned on;) method to record data

```
/// <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 CANId;
    try
    {
        portName = (string)portComboBox.SelectedItem;
        baudrate = (int)baudComboBox.SelectedItem;
        CANId = byte.Parse(ModbustextBox.Text.Replace("0x", ""),
        System.Globalization.NumberStyles.HexNumber);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

    // Do not open again
    if (HWT9073CAN.IsOpen())
    {
        return;
    }

    // Turn on the device
    try
    {
        HWT9073CAN.SetPortName(portName);
        HWT9073CAN.SetBaudrate(baudrate);
```

```
            HWT9073CAN.SetCANId(CANId);
            HWT9073CAN.Open();
            // Implement logging data events
            HWT9073CAN.OnRecord += HWT9073CAN_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="HWT9073CAN"></param>
private void HWT9073CAN_OnRecord(HWT9073CAN HWT9073CAN)
{
    string text = GetDeviceData(HWT9073CAN);
    Debug.WriteLine(text);
}
```

Set up the sensor

The sensor can be operated by the method of HWT9073CAN
HWT9073CAN.UnlockReg() Send unlock register command
HWT9073CAN.AppliedCalibration() Sends the addition calibration command
HWT9073CAN.StartFieldCalibration() Send start field calibration command
HWT9073CAN.EndFieldCalibration() Send end field calibration command
HWT9073CAN.SetReturnRate() Send the command to set the return rate
HWT9073CAN.SetBandWidth() Send command to set bandwidth
HWT9073CAN.SendProtocolData() Send other commands

SET ADDRESS

Set the sensor address by calling the HWT9073CAN.SetCANId() 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 (HWT9073CAN.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        HWT9073CAN.UnlockReg();
        byte CANId = byte.Parse(ModbustextBox.Text.Replace("0x", ""),
        System.Globalization.NumberStyles.HexNumber);
        HWT9073CAN.SetCANId(CANId);
        // The following two lines are equivalent to the above, and it is recommended to
        use the above
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x1A, 0x50, 0x00, });
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

ACCELERATION CALIBRATION

Acceleration calibration of the sensor by calling the HWT9073CAN.AppliedCalibration() method

```
/// <summary>
/// Acceleration calibration
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void appliedCalibrationButton_Click(object sender, EventArgs e)
{
    if (HWT9073CAN.IsOpen() == false)
    {
        return;
    }

    try
    {
        // Unlock registers and send commands
        HWT9073CAN.UnlockReg();
        HWT9073CAN.AppliedCalibration();
```

```
            // The following two lines are equivalent to the above, and it is recommended to
            use the above
            //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
            //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x01, 0x01, 0x00 });
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
}
```

MAGNETIC FIELD CALIBRATION

Perform magnetic field calibration on the sensor by calling the HWT9073CAN.StartFieldCalibration() method and the HWT9073CAN.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 (HWT9073CAN.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        HWT9073CAN.UnlockReg();
        HWT9073CAN.StartFieldCalibration();
        // The following two lines are equivalent to the above, and it is recommended to
        use the above
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x01, 0x07, 0x00 });
        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 (HWT9073CAN.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        HWT9073CAN.UnlockReg();
        HWT9073CAN.EndFieldCalibration();
        // The following two lines are equivalent to the above, and it is recommended to
        use the above
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x01, 0x00, 0x00 });
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

SET THE RETURN RATE

Set the return rate of the sensor by calling HWT9073CAN. SetReturnRate() method.

```
/// <summary>
/// Set the return rate 10Hz
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void returnRate10_Click(object sender, EventArgs e)
{
    if (HWT9073CAN.IsOpen() == false)
    {
        return;
    }
    try
    {
```

```
            // Unlock registers and send commands
            HWT9073CAN.UnlockReg();
            HWT9073CAN.SetReturnRate(0x06);
            // The following two lines are equivalent to the above, and it is recommended to
use         the above
            //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
            //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x03, 0x06, 0x00 });
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
}
```

SET BANDWIDTH

Set the bandwidth of the sensor by calling HWT9073CAN. 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 (HWT9073CAN.IsOpen() == false)
    {
        return;
    }
    try
    {
        // Unlock registers and send commands
        HWT9073CAN.UnlockReg();
        HWT9073CAN.SetBandWidth(0x04);
        // The following two lines are equivalent to the above, and it is recommended to
use the above
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x1F, 0x04, 0x00 });
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Read sensor register

The sensor register can be read through the HWT9073CAN.SendReadReg() method, or the HWT9073CAN.SendProtocolData() method can be used
After sending the read command, the register value will be saved in HWT9073CAN, and the register data needs to be obtained through HWT9073CAN.GetDeviceData().

```
/// <summary>
/// read 03 register
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void readReg03Button_Click(object sender, EventArgs e)
{
    if (HWT9073CAN.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.
        HWT9073CAN.SendReadReg(0x03, waitTime);
        // The following two lines are equivalent to the above, and it is recommended to
        use the above
        //HWT9073CAN.SendProtocolData(new byte[] { 0xff, 0xaa, 0x27, 0x03, 0x00 },
        waitTime);

        string reg03Value = HWT9073CAN.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 SetReturnRate(byte rate)	Set return rate	rate: the return rate to be set	void
void SetBandWidth(byte band)	Set bandwidth	Band: the bandwidth to 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 SetCANId(byte CANId)	Specify address	CANId: CAN address	void