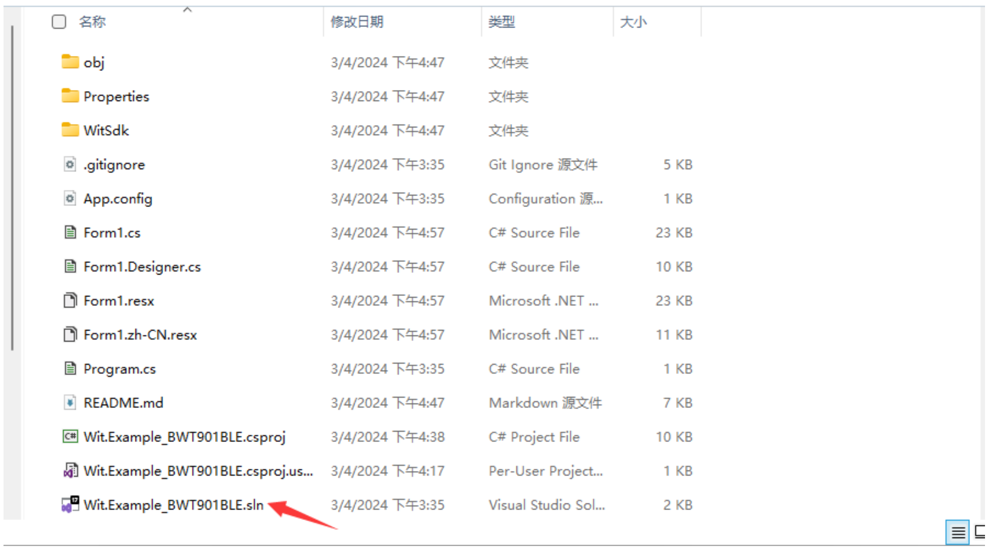
# C#_SDK_Quick Start

## Routine Description



| 名称 | 修改日期 | 类型 | 大小 |
|---|---|---|---|
| obj | 3/4/2024 下午4:47 | 文件夹 | |
| Properties | 3/4/2024 下午4:47 | 文件夹 | |
| WitSdk | 3/4/2024 下午4:47 | 文件夹 | |
| .gitignore | 3/4/2024 下午3:35 | Git Ignore 源文件 | 5 KB |
| App.config | 3/4/2024 下午3:35 | Configuration 源... | 1 KB |
| Form1.cs | 3/4/2024 下午4:57 | C# Source File | 23 KB |
| Form1.Designer.cs | 3/4/2024 下午4:57 | C# Source File | 10 KB |
| Form1.resx | 3/4/2024 下午4:57 | Microsoft .NET ... | 23 KB |
| Form1.zh-CN.resx | 3/4/2024 下午4:57 | Microsoft .NET ... | 11 KB |
| Program.cs | 3/4/2024 下午3:35 | C# Source File | 1 KB |
| README.md | 3/4/2024 下午4:47 | Markdown 源文件 | 7 KB |
| Wit.Example_BWT901BLE.csproj | 3/4/2024 下午4:38 | C# Project File | 10 KB |
| Wit.Example_BWT901BLE.csproj.us... | 3/4/2024 下午4:17 | Per-User Project... | 1 KB |
| Wit.Example_BWT901BLE.sln | 3/4/2024 下午3:35 | Visual Studio Sol... | 2 KB |

Directions:

Install Visual Studio 2019 or higher on your computer and select the sln solution to enter

## Download the example

Go to the link below to download the example

https://github.com/WITMOTION/WitBluetooth_BWT901BLE5_0/tree/main/Windows_C%23

## Noun introduction

Bluetooth 5.0 protocol: This is the protocol used by Witt smart Bluetooth 5.0 sensors. The protocol stipulates that the sensor returns data packets starting with 55, and the host computer sends data packets starting with FF AA.

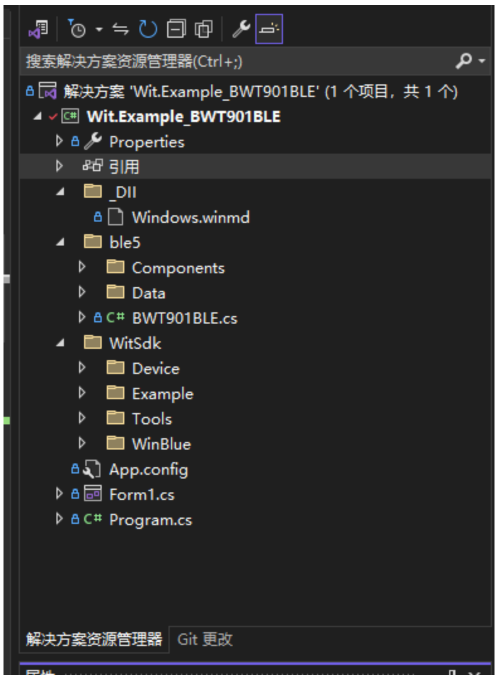
# Routine Introduction

This example introduces how to use C# to develop a host computer to connect to a Bluetooth 5.0 protocol sensor, receive sensor data and communicate with the sensor;

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

## Routine Directory

The routine project directory is as follows



```
搜索解决方案资源管理器(Ctrl+;)
解决方案 'Wit.Example_BWT901BLE' (1 个项目，共 1 个)
  Wit.Example_BWT901BLE
    Properties
    引用
    _Dll
      Windows.winmd
    ble5
      Components
      Data
      BWT901BLE.cs
    WitSdk
      Device
      Example
      Tools
      WinBlue
    App.config
    Form1.cs
    Program.cs
解决方案资源管理器  Git 更改
```

Folder Description:

_Dll: The sample program Bluetooth connection dependency library. You need to import the files in it into the project reference.

ble5: BWT901BLE5.0 device model

WitSdk: Wit sample program dependency package

Form1: sample program entry, main interface, this sample only has this interface

# Open Search

The BWT901BLE object represents the BWT901BLE device in the program. You can use it to communicate with the device. When searching for a device, call bluetoothManager.StartScan() to start the search. When opening a device, you need to specify the Bluetooth address of the Bluetooth sensor. After specifying it, call the BWT901BLE.Open() method.

```
/// <summary>
/// Start searching
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void startScanButton_Click(object sender, EventArgs e)
{
// Clear the found device
FoundDeviceDict.Clear();

// Close the previously opened device
for (int i = 0; i < FoundDeviceDict.Count; i++)
{
var keyValue = FoundDeviceDict.ElementAt(i);
BWT901BLE bWT901BLE = keyValue.Value;
bWT901BLE.Close();
}

// Let the Bluetooth manager start searching for devices
WitBluetoothManager bluetoothManager =
WitBluetoothManagerHelper.GetWitBluetoothManager();
bluetoothManager.OnDeviceFound += new
WitBluetoothManager.OnDeviceFoundHalder(OnFoundDevice);
bluetoothManager.OnDeviceStatu += new
WitBluetoothManager.OnDeviceStatuHalder(OnDeviceStatu);
bluetoothManager.StartScan();
}

/// <summary>
/// This method will be called back when a Bluetooth device is found
/// </summary>
/// <param name="macAddr"></param>
/// <param name="sName"></param>
/// <param name="dType"></param>
/// <param name="mType"></param>
private void OnFoundDevice(string macAddr, string sName, int dType, int mType)
{
// If a Bluetooth device starting with WT is found
if (sName != null && sName.Contains("WT"))
{
// If this device is newly found
if (FoundDeviceDict.ContainsKey(macAddr) == false)
{
BWT901BLE bWT901BLE = new BWT901BLE();
// Specify the connected Bluetooth MAC code
bWT901BLE.SetMacAddr(macAddr);
// Set the device name
bWT901BLE.SetDeviceName(sName);
FoundDeviceDict.Add(macAddr, bWT901BLE);
// Open this device
bWT901BLE.Open();
bWT901BLE.OnRecord += BWT901BLE_OnRecord;
}
}
}
```

## Close Search

To stop searching, call the bluetoothManager.StopScan() method.

```
/// <summary>
/// Stop searching
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void stopScanButton_Click(object sender, EventArgs e)
```

```
{
// Let the Bluetooth manager stop searching
WitBluetoothManager bluetoothManager =
WitBluetoothManagerHelper.GetWitBluetoothManager();
bluetoothManager.StopScan();
}
```

## Receiving sensor data

### Get the data

The BWT901BLE object will automatically solve the sensor data and save it on itself. The sensor data can be obtained through the BWT901BLE.GetDeviceData() method. BWT901BLE.GetDeviceData() needs to pass in a key to obtain the sensor data. Please check the key you need to use in the example. The key is saved in the WitSensorKey class.

```
/// <summary>
/// Get device data
/// </summary>
private string GetDeviceData(BWT901BLE BWT901BLE)
{
StringBuilder builder = new StringBuilder();
builder.Append(BWT901BLE.GetDeviceName()).Append(" \n");
// Acceleration
builder.Append("AccX").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKey.
AccX)).Append("g \t");
builder.Append("
AccY").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKey.AccY)).Append("g
\t");
builder.Append("AccZ").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKey.
AccZ)).Append("g \n");
// Angular velocity
builder.Append("GyroX").Append
(":").Append(BWT901BLE.GetDeviceData(WitSensorKey.AsX)).Append("°/s \t");
builder.Append("GyroY").Append(":").Append(BWT901BLE.GetDeviceData
(WitSensorKey.AsY)).Append("°/s \t");
builder.Append("GyroZ").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKey
.AsZ)).Append("° /s \n");
// Angle
builder.Append("AngleX").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKe
y.AngleX)).Append("° \t");
builder.Append("AngleY").Append("
:").Append(BWT901BLE.GetDeviceData(WitSensorKey.AngleY)).Append("° \t");
builder.Append("AngleZ").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKe
y.AngleZ )).Append("° \n");
// Magnetic field
builder.Append("MagX").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKey.
HX)).Append("uT \t" );
builder.Append("MagY").Append(":").Append(BWT901BLE.GetDeviceData(WitSensorKey.
HY)).Append("uT \t");
builder.Append("MagZ").Append(":
").Append(BWT901BLE.GetDeviceData(WitSensorKey.HZ)).Append("uT \n");
// Version number
builder.Append("VersionNumber").Append(":").Append(BWT901BLE.GetDeviceData(
WitSensorKey.VersionNumber)).Append("\n");
return builder.ToString();
}
```

## Recording Data

The sensor data can be obtained through the BWT901BLE object, but usually the host computer needs to record the sensor data. BWT901BLE has an OnRecord event that will notify you when to record data. The OnRecord event can be implemented when the device is turned on; then the data can be recorded by coordinating the BWT901BLE.GetDeviceData() method

```
/// <summary>
/// This method will be called back when a Bluetooth device is found
/// </summary>
/// <param name="macAddr"></param>
/// <param name="sName"></param>
/// <param name="dType"></param>
/// <param name="mType"></param>
private void OnFoundDevice(string macAddr, string sName, int dType, int mType)
{
// If a Bluetooth device starting with WT is found
if (sName != null && sName.Contains("WT"))
{
// If this device is newly found
if (FoundDeviceDict.ContainsKey(macAddr) == false)
{
BWT901BLE bWT901BLE = new BWT901BLE();
// Specify the Bluetooth MAC code for the connection
bWT901BLE.SetMacAddr(macAddr);
// Set the device name
bWT901BLE.SetDeviceName(sName);
FoundDeviceDict.Add(macAddr, bWT901BLE);
// Open this device
bWT901BLE.Open();
bWT901BLE.OnRecord += BWT901BLE_OnRecord;
}
}
}

/// <summary>
/// This will be called when the sensor data is refreshed. You can record the data here
. /// </summary>
/// <param name="BWT901BLE"></param>
private void BWT901BLE_OnRecord(BWT901BLE BWT901BLE)
{
string text = GetDeviceData(BWT901BLE);
Debug.WriteLine(text);
}
```

## Setting up the sensor

The sensor can be operated by BWT901BLE method

BWT901BLE.UnlockReg() Sends the unlock register command

BWT901BLE.AppliedCalibration() Sends an applied calibration command

BWT901BLE.StartFieldCalibration() Sends a command to start magnetic field calibration

BWT901BLE.EndFieldCalibration() Sends the command to end magnetic field calibration

BWT901BLE.SendProtocolData() Send other commands

## Calibration

Calibrate the sensor by calling the BWT901BLE.AppliedCalibration() method

```
/// <summary>
/// Add calibration
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void appliedCalibrationButton_Click(object sender, EventArgs e)
{
// All connected Bluetooth devices are calibrated
for (int i = 0; i < FoundDeviceDict.Count; i++)
{
var keyValue = FoundDeviceDict.ElementAt(i);
BWT901BLE bWT901BLE = keyValue.Value;

if (bWT901BLE.IsOpen() == false)
{
return;
}

try
{
// Unlock the register and send the command
bWT901BLE.UnlockReg();
bWT901BLE.AppliedCalibration();
// The following two lines are equivalent to the above, and it is recommended to use the
above
//bWT901BLE.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
//bWT901BLE.SendProtocolData(new byte[] { 0xff, 0xaa, 0x01, 0x01, 0x00 });
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}
}
```

## Magnetic field calibration

Calibrate the sensor magnetic field by calling the BWT901BLE.StartFieldCalibration() method and the BWT901BLE.EndFieldCalibration() method

```
/// <summary>
/// Start magnetic field calibration
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void startFieldCalibrationButton_Click(object sender, EventArgs e)
{
// Start magnetic field calibration for all connected Bluetooth devicesfor
(int i = 0; i < FoundDeviceDict.Count; i++)
{
var keyValue = FoundDeviceDict.ElementAt(i);
BWT901BLE bWT901BLE = keyValue.Value;

if (bWT901BLE.IsOpen() == false)
{
return;
}
try
{
```

```
// Unlock registers and send commandsbWT901BLE.UnlockReg
();
bWT901BLE.StartFieldCalibration();
// The following two lines are equivalent to the above, it is recommended to use the above
//bWT901BLE.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
//bWT901BLE.SendProtocolData(new byte[] { 0xff, 0xaa, 0x01, 0x07, 0x00 });
MessageBox.Show("Start magnetic field calibration. Please rotate around the sensor's XYZ
axes once each. After the rotation, click [End magnetic field calibration]");
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}
}

/// <summary>
/// 末磁电开关
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void endFieldCalibrationButton_Click(object sender, EventArgs e)
{

// End magnetic field calibration for all connected Bluetooth devicesfor
(int i = 0; i < FoundDeviceDict.Count; i++)
{
var keyValue = FoundDeviceDict.ElementAt(i);
BWT901BLE bWT901BLE = keyValue.Value;

if (bWT901BLE.IsOpen() == false)
{
return;
}
try
{
// Unlock the register and send the command
bWT901BLE.UnlockReg();
```

```
bWT901BLE.EndFieldCalibration();
// The following two lines are equivalent to the above, and it is recommended to use the
above
//bWT901BLE.SendProtocolData(new byte[] { 0xff, 0xaa, 0x69, 0x88, 0xb5 });
//bWT901BLE.SendProtocolData(new byte[] { 0xff, 0xaa, 0x01, 0x00, 0x00 });
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}
}
```

## More

For other operations, please refer to the sensor manual.

## Read sensor registers

You can read the sensor registers using the BWT901BLE.SendReadReg() method or the BWT901BLE.SendProtocolData() method.

After sending the read command, the register value will be saved in BWT901BLE. You need to use BWT901BLE.GetDeviceData() to get the register data.

```
/// <summary>
/// Read register 03
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void readReg03Button_Click(object sender, EventArgs e)
{
string reg03Value = "";
// Read register 03 of all connected Bluetooth devicesfor
(int i = 0; i < FoundDeviceDict.Count; i++)
{
```

```
var keyValue = FoundDeviceDict.ElementAt(i);
BWT901BLE bWT901BLE = keyValue.Value;

if (bWT901BLE.IsOpen() == false)
{
return;
}
try
{
// Waiting timeint
waitTime = 3000;
// Send a read command and wait for the sensor to return data. If the data is not read, the
waitTime can be extended, or read several timesbWT901BLE.SendReadReg
(0x03, waitTime);

// The following line is equivalent to the above. It is recommended to use the above
//bWT901BLE.SendProtocolData(new byte[] { 0xff, 0xaa, 0x27, 0x03, 0x00 }, waitTime);

// Get the values of all connected Bluetooth devices
reg03Value += bWT901BLE.GetDeviceName() + "The value of register 03 is:" +
bWT901BLE.GetDeviceData("03") + "\r\n";
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}
MessageBox.Show(reg03Value);
}
```

## BWT901BLE API

| method | illustrate | Parameter Introduction | Return Value |
|---|---|---|---|

| | | | |
|---|---|---|---|
| void SetMacAddr(string macAddr) | Set the Bluetooth address to open | macAddr: Bluetooth address | void |
| void SetDeviceName(string DeviceName) | Set the device name | DeviceName: device name | void |
| void Open() | Open the device | none | void |
| bool IsOpen() | Is the device turned on? | none | Returns whether it is open Open: true Close: false |
| void Close() | Turn off the device | none | void |
| void SendData(byte[] data, out byte[] returnData, bool isWaitReturn, int waitTime, int repetition) | Sending Data | data: data to be sent<br><br>returnData: data returned by the sensor<br><br>isWaitReturn: Whether the sensor needs to return data<br><br>waitTime: Waiting time for the sensor to return data, in milliseconds, default | void |

| | | | |
|---|---|---|---|
| | | is 100ms<br><br>repetition: number of repetitions | |
| void SendProtocolData(byte[] data) | Sending data with protocol | data: data to be sent | void |
| void SendProtocolData(byte[] data, int waitTime) | Send data with protocol and specify waiting time | data: data to be sent<br><br>waitTime: waiting time | void |
| void SendReadReg(byte reg, int waitTime) | Send a command to read the register | reg: command to be sent<br><br>waitTime: waiting time | void |
| void UnlockReg() | Unlock Register | none | void |
| void SaveReg() | Save registers | none | void |
| void AppliedCalibration() | Calibration | none | void |
| void StartFieldCalibration() | Start magnetic field calibration | none | void |
| void EndFieldCalibration() | End magnetic field calibration | none | void |
| void SetReturnRate(byte rate) | Set the return rate | rate: The return rate to be set | void |
| string GetDeviceName() | Get device | none | Returns the |

| | | | |
|---|---|---|---|
| | name | | device name |
| string GetDeviceData(string key) | Get Key value data | key: data key value | Return data value |